Critical Path Search and Network Prediction Acceleration

Chenyu Yang and Xing Zhao
Shanghai Jiao Tong University

The widespread use of machine learning has made neural networks popular, which has led to some areas of demand for neural network speed. There are many ways to speed up the neural network. The main discussion in this paper is to accelerate the neural network through critical path prediction. There has been a lot of related work to explore critical paths through control gates or some other neuron's criteria. We also implemented some pruning-based methods to find critical paths, and did some research on these critical paths, hoping to find valuable features and use them to accelerate the neural network prediction process.

1. INTRODUCTION

Critical path in neural network has gain great interests recently. Some subset of the neural networks are found to be able to affect drastically of the result of the neural network of some specific input class. [Zhou et al. 2018] found that by ablating a neuron representing the bed, the neural network performs poorly on input sets containing bet like hotel and bedroom. Thus, people begin to try to find critical paths and make use of them.[Qin et al. 2018] tries to prune redundant filters on each "critical path" to get a slim network. [Wang et al. 2018] tries to utilize their critical path as an imbedding of the input, and from which they perform dimension reduction work, and detect adversarial samples.

Note that the "critical path" that each the previous work defined are different of each other. But they all based on the same observation that some neurons contributes more to the result of the neural network on a single input. In this project, we take advantages of this observation and try to use "critical path" to make the inference process of neural network faster. Here the "critical path" we use in this project is the subset of the neural network, that have significant effects on a specific class of input. And the accuracy for the neural network discriminating the class do not degrade very much when all the other neurons outside of the "critical path" are pruned out. What's more, we hope that our "critical path" can have some characters that enables us to decide which class the input likely to belongs to during the inference, based on only the information of a feature map in middle layers. With this kind of "critical path", we can guess the flow of the activation at the runtime, thus omits the calculation of the kernels that do not contribute much to the result, and saves the time and energy.

We planned our work to be carried out in the following procedure:

First, we consult traditional pruning methods and other methods of distilling critical path to find the critical path out. There are plenty of successful pruning methods, and we can use their methods with slight adaption, changing from pruning for a subset of network for whole input space to pruning for a subset of network for a specific class of input.

Second, we study the characteristics of the critical path. That is to say, to check whether they are very different from each other on different input class, whether the activation on the critical path can give us enough information for us to safely omit the calculation of some filters, and what metrics can we rely on to guess the critical path at the runtime.

Third, we design algorithms to take advantage of the characteristics about the critical path we've found. The algorithms have to be safe enough to preserve the accuracy and simple enough to make sure that the overhead is no greater than the time saved by omitting the neurons not on the critical path.

2. BACKGROUND AND RELATED WORKS

Recently researchers found that the neurons in higher layers of neural networks are usually specialized to encode information relevant to a subset of classification targets. For example, by ablating certain neurons in higher layers, significant accuracy degradation of specific classes can be observed[Yu et al. 2018]. For these neurons, we think they play a crucial discriminating role in the network to distinguish specific classes. We call these sets of neurons the critical paths for a certain classes, which means for each class, the critical paths will have some intersection differences. This feature has attracted some attentions in the field and there have been some researches about this topic.

The methods can be roughly categorized into two routes. The first one is to add a gate after each neuron in the network. It discovers the critical nodes on the data routing paths during network inferring prediction for individual input samples by learning associated control gates for each layers output channel. The routing paths can, therefore, be represented based on the responses of concatenated control gates from all the layers, which reflect the networks semantic selectivity regarding to the input patterns and more detailed functional process across different layer levels[Wang et al. 2018]. The schematic is shown in fig1. In this method, the core operation is to further train the control gate so that the critical path can be extracted from the control gate obtained after training, while the output of the new network on the data set is as close as possible to the original model output. To achieve this aim, a method named Distillation Guided Routing(DGR) is developed. In this method, the control gates are trained according to a loss function based on the difference between the output of the original network and the network with control gates. Specifically, the optimization objective for all the control gates Λ is

$$\begin{aligned} \min_{\Lambda} \quad & \mathcal{L}(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}; \boldsymbol{\Lambda})) + \gamma \sum_k |\boldsymbol{\lambda}_k|_1 \\ s.t. \quad & \boldsymbol{\lambda}_k \geq 0, k = 1, 2, ..., K, \end{aligned}$$

where $\mathcal{L}$ is the cross entropy loss between the original full model's prediction probability $f_\theta(\boldsymbol{x}) = [p_1, p_2, ..., p_m]$and the new prediction probability $f_\theta(\boldsymbol{x}, \boldsymbol{\Lambda}) = [q_1, q_2, ..., q_m]$,which is $\mathcal{L} = \sum_i^m -p_i \log q_i$, where m is the category number, and γ is the balanced parameter.

The other method use evaluation criterion based on the activation value such as Mean Absolute Activation, Contribution Index and AM Visualization to identify critical paths based on the neuron contribution significance to specific classes[Yu et al. 2018]. Similarly, there is another work focusing on the neuron ablation.[Wang et al. 2018] shows some methods based on the unit ablation or other attributes such as L1 form, Class Correlation, Class Selectivity and

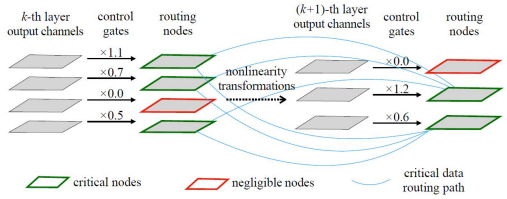

Fig. 1. **gate based critical path** The control gates are multiplied to the layers output channel-wise, resulting in the actual routing nodes. In this demonstration, we identify those nodes whose responses of the control gates are larger than 0. The layer-wise routing nodes are linked together to compose the routing paths.

Concept Alignment to identify the individual units in the neural network.

3. MOTIVATION

Nowadays, the popularity of artificial intelligence has made application problems in various fields unprecedented. Among them, the generation and development of neural networks play an irreplaceable role. It has the ability to fit various functional relationships, and the corresponding functional relationships can be extracted from the training samples and training tags, and expressed by network weights. However, in some complex functional relationships, large-scale networks are often used, which contain many weighting parameters. As mentioned in the previous section, only a part of the parameters in such a network are crucial for the identification of specific categories, which leads to a large part of the parameter redundancy in the network. They increase the amount of computation during the use of the network, but do not greatly improve the accuracy of network identification. This has led to some extent in some applications, the speed and size of the network is not satisfactory. We've mentioned the critical path in the previous section, which plays a major role in identifying a specific class. Perhaps we can use this feature to predict the judgment tendency of network recognition results in real time in the process of large-scale neural network identification to avoid unnecessary forward calculation, so as to improve the recognition speed of the network while maintaining network recognition accuracy.

4. PROBLEM FORMULATION

As identified in the title, our problem is mainly consists of the identification of critical paths and the acceleration during the predicting process.

4.1 Critical Path Search

With a well-trained neural network and some tagged data sets, we need to find network nodes that cannot be removed in a certain kind of class identification and generate critical paths of this type based on these nodes. The definition of critical path is also a problem need to be solved in the implement process.

4.2 Prediction Acceleration

After getting all kinds of critical paths, we need to analyze the characteristics of these critical paths and the relationship between the critical paths of each class. Based on these findings, we propose a real-time prediction of the possible outcomes of the classification based on the trend of some criteria in the network during the network forward prediction process and use the critical paths to select a part of the network to calculate and output the prediction results. This process can be implemented by continually selecting combinations of critical paths for each class, or discarding critical paths of certain classes from existing networks.

5. PROPOSED METHODS

The first step of using critical path to make acceleration for the neural network inference and training is to find it out, and from which several properties can be examined.

5.1 Method for Picking Critical Path

5.1.1 *Picking Critical Path with Activation and Contribution index map.*

5.1.1.1 *Iterative pruning process.* We transplanted a method for pruning the neural network to prune the critical path out with some statistical information collected in the network running process. The original pruning method contains the following procedure:

(1) run the neural network several times with the input from the data set
(2) rank the filters of each layer with the information collected during the running process
(3) prune out the filters that have low ranking
(4) fine tune
(5) go back to step 1

Our procedure is quite similar from the original one. As what we want are critical paths of certain output classes, we have to collect the statistical information specifically toward one kind of input. So we can achieve that by feeding only one kind of input and run the network. Our method contains the following procedure:

(1) run the neural network several times with the input from the dataset of one specific input class
(2) rank the filters of each layer with the information collected during the running process
(3) prune out the filters that have low ranking
(4) go back to step 1

Note that we do not have a fine tune step in this procedure for the following reasons:

(1) Comparing with the original algorithm, where the algorithm trying to find is a new neural network with high accuracy and small structure, the target of our method is to find out the subsets that make great contribution to some classes of input without changing the weights of the original network.
(2) We do not have accuracy restriction for our pruned network, omitting the find tuning process saves the time and effort.

5.1.1.2 *Ranking Metrics.* The ranking metrics of the filters is derived from the first-order Tyler expansion, which utilize the backward-propagation gradients and the activation value. Given a layer l with I_l filters, the neural network output could be formulated by any layer l's output feature maps in the following format:

$$Z(F, A^l) = F^L(...\alpha(F^{l+1} * A^l + B^{l+1})...) + b^L \quad (1)$$

where F is the filter weight and bias in every layer. A^l is the output feature maps (activation maps) of layer l . The approximate function of Z can be calculated by the first order Tyler-expansion.

$$Z(A^l) = \frac{\partial Z(A^l)}{\partial A^l}.A^l + O(A^{l^2}) \tag{2}$$

As the filter's output feature map is about zero, so the Δ in the Tyler expansion equation can be A^l . Thus, before pruning, each filter $F_i^{(c)}$ in the class of input C is firstly ranked by the contribution index

$$I(F_i^c) = \frac{1}{N}\sum_{n=1}^{N} || \frac{\partial Z(F, A^l)}{\partial A_i^l(x_n)} * A^l || \tag{3}$$

Where the $Z(F, A^l)$ is the neural network output loss of a sample image n of class c, and the $A_i^l(x_n)$ is the feature map of filter i for each test image n in input class c.

Note that in this formula, the A^l term can be regarded as importance of the filter given all the networks before the layer l, that is, the topology and the weights which made the activation value. The term $\frac{\partial Z(F, A^l)}{\partial A_i^l(x_n)}$ can be regarded as how importance the filter is with the connection of the network behind the layer l, that is, whether the later network serve to amplify or diminish the effect of filter F on input class c.

5.1.2 *Distilling critical path with ADMM pruning.*

5.1.2.1 *ADMM pruning and original formulation.* The work combined the need for accuracy and the sparsity in the loss function. The sparsity requirement is not embodied a regulation function like Lasso regulation, but by a discrete defined function to force the weight of the layer has numbers that not equal to zero less than some certain number. Then the work use ADMM to solve the optimization problem involving non-differentiable terms.

The problem can be written specifically as following:

$$\underset{\{W_i\},\{b_i\}}{\text{minimize}}\, f(\{W_i\},\{b_i\}) \text{subject to } \; card(W_i) \leq l_i, i = 1, ..., N, \tag{4}$$

Where the $\{W_i\}$ represents the collection of the collections of weights W_i, in other words, all the weights. Similarly, $\{b_i\}$ represents all the bias. f is a loss function, specifically:

$$f(\{W_i\},\{b_i\}) = -\frac{1}{t}\sum_{i=1}^{t} \log \frac{e^{s_{yj}j}}{\sum_{i=1}^{k} e^{s_{ij}}} + \lambda \sum_{i=1}^{N} ||W_i||_F^2 \tag{5}$$

The second term is the L_2 weight regularization, providing the regularize function which cannot be provided by the discrete $card$ restriction.

To make the formulation fit to ADMM, the work takes a step of transformation, and get:

$$\underset{\{W_i\},\{b_i\}}{\text{minimize}}\, f(\{W_i\},\{b_i\}) + \sum_{i=1}^{N} g_i(Z_i) \text{subject to } \; W_i = Z_i, i = 1, ..., N, \tag{6}$$

The function g is simple penalty for the weight sets that do not meet the sparsity requirement:

$$g_i(W_i) = \begin{cases} 0 & \text{if } card(w_i) \leq l_i \\ +\infty & \text{otherwise} \end{cases} \tag{7}$$

This term ensures that the solution is sparsity. While it is non-differentiable, it is demonstrated that such problems lend themselves well to the application of ADMM. By rewriting the above problem in ADMM form as the previous function 6, the augmented Lagrangian form can be further derived and process of ADMM can be carried out.

5.1.2.2 *Our method using ADMM.* We transplanted the method discussed above to make it distill critical path. The process is quite straight forward. As what we are interested in are only the subset of the network which have strong effect about the result on one single input class. That is to say, we want to distill out the sub-networks with one-vs-all functionality, so we changed the output space from N dimension (the number of input classes) to 2 dimension, whether or not the input belongs to the class of interest. We do this by changing the last FC layer of the network into a new trainable FC layer with 2 output dimension. We first freeze the former convolutional layers and train the fully connected layer, and then do the ADMM process. After this , we can still fine tune the last layer to preserve the accuracy. Pruning the last FC layers is a good idea that can also be used in the method of pruning with Tyler expansion, we will compare the two results in the later sections.

Thus, the overall process for our ADMM pruning process is as following:

(1) Change the last layer of the network into fully connected layer with 2 output dimension
(2) Freeze the convolutional layers and train(fine-tune) the last layer
(3) Do ADMM process to get the sparse weight sets
(4) go to step 2.

As the ADMM methods aim at the accuracy and sparsity, it surely have to change the weights of the convolutional layer. what's more, as the pruning method works at a fine-grain size–pruning at each single weights–we can only assume that the filter with all weights equal to 0 are pruned out. As the method can change a great portion of weights to 0, the number of filters with all weights equal to zero is significant enough to prune the critical path out.

5.2 Acceleration method

As we described before, we reason that we look into the critical path is that we hope the characteristics of those critical paths can help us in the network acceleration. So, suppose we have found out the critical paths based on previous methods, we can achieve acceleration by taking advantage of the critical paths. Unfortunately, as we have not finish the first part, we can only develop our acceleration algorithms based on assumptions., and hope that the previous methods can verify the assumptions. To make the acceleration work, we have the following assumptions:

(1) The neural network have large redundancy when doing inference for a specific class. Cutting off a portion of the networks does not affect the neural network discriminate accuracy of a certain class greatly.
(2) The activation in the network are correlated. That is, there exists some "pattern"s in the form of critical path. If the neurons at the previous layers of the critical path have large activation, the neurons at the later layers of the critical path are likely to have large activation.
(3) The activation on the previous layer can provide enough information for us to determine which critical path the flow of activation will take, and this process of determination should

be simple enough, no larger than the acceleration we can get from only running on the critical path.

The first assumption makes the acceleration possible, that is to say, we can run only a subset of the network to get the result as accurate as that on a whole network. Note that the redundancy here is the redundancy relating a specific class. Even a well-pruned network that can handle normal input reference tasks should still have redundancy considering only one kind of input.

The second assumption forms the critical path we are looking for, and the third assumption ensures that the it is possible for us to decide the critical path online. Because the information we have at the running time is only the activation maps, we cannot hope to decide which the input might be with very complex metrics like derivation or very complex procedure like mean squaring.

proposed algorithm With the above assumptions, we propose the algorithm as following: Suppose for a convolution layer i, the number of input feature maps and the number of output feature maps are M and N. Then for this layer i, we maintain a Boolean matrix C_i, with shape $M * N$,which contains the structure of the critical path.

In this way, at the running time, when it is about to calculate the feature map f_i of layer i according to feature map f_{i-1}, we do the following steps at first.

(1) Make a Boolean vector I of shape $(M,)$, each number represent whether the feature map f_{i-1} is activated. By "activated", it means the sum of activation on the activation map is larger than a threshold value.
(2) Calculate $J = I * C$.
(3) Change the vector J to be a Boolean vector, each number represents whether J is larger than some threshold.
(4) Do the inference step with only the feature maps that has 1 in their corresponding Boolean vector.

6. EXPERIMENTS

6.1 Network and data-set

We chose the VGG to be the target network, and we train and test it on MNIST data set. Later we moved to cifar10 data set, for the features in MNIST might not be complex enough for a large network like VGG. The training process has large likely hood of overfitting.

6.2 Test the pattern assumption of network

To test our assumption that the activation at previous layers have some kind of "pattern", we made a clustering experiment about the activation, and use ground truth class labels to test whether the activation have clear patterns. We did PCA about the activation of each layer of the network. The result is shown in fig 2.

From the figures we can find that the patterns of the activation are not that ideal. From the figure, they have large portions coincide with each other. However, the 2D embedding is not sufficient to prove the difficulties of distilling out the critical path. What we are looking for is a set of ten sub networks. Thus it is entirely possible that the activation can be clearly divided into 10 clusters. However, for the sake of visual clearness, we embedded them into two clusters instead of ten.

6.3 critical path pruning

6.3.1 *with Tyler pruning methods.* We use the methods described in 5.1.1 get ten sets of sub networks. And tested the precision and recall of the pruned networks. In the following graphs,

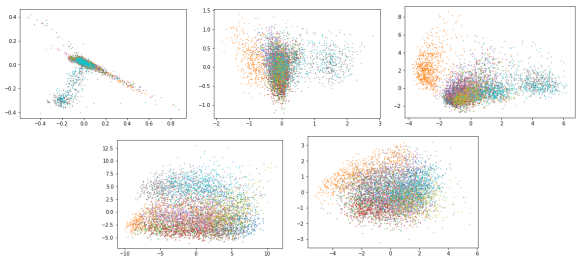

Fig. 2. **PCA clustering of activation.** The figures are the PCA results for the fist layer, the Third layer, the fifth layer, the 8th layer, and the 13th layer

each of the figure have x-axis represents the percentage of the network pruned, and the y-axis represents the accuracy of the precision and recall of the networks. As the sub network cares about only one class of input, so the problem became an one-vs-all problem, and that is the reason the precision and recall is examined.

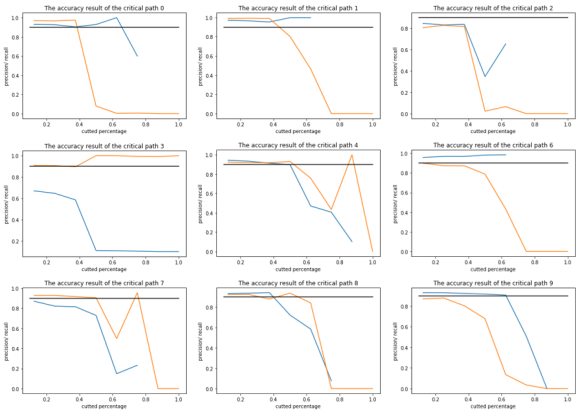

Fig. 3. **pruned network accuracy performance** The accuracy curves of the sub networks of different classes. The axis are the pruned percentage and the value of precision and recall. The yellow lines represent recall and the blue lines represent precision. For the sake of brevity the provide only 9 figures

From the pruning results we can find that the accuracy performance drops quickly as we prune the network without fine-tuning. So we choose the sub-network when about fifty percent of networks are pruned as the critical path found in this method. Another interesting observation is that the recall percentage bounced back in the pruned results of class 4 and 7. We further repeated this experiment and found that this kind of observation is quite common, although happens at other . Tshes of inputssis means

that some of the filters pruned in this way is responsible for deciding what kind of the input do not belongs to the target class. It might be because that this kind of neuron have large negative effect directly on the output dimension representing the class, or because that this kind of neurons are positive related to other classes, and they suggest that some inputs do not belongs to the input class by suggesting that it belongs to some other input classes. It needs more experiments for us to discover which is the true reason for these observation.

6.3.2 *with ADMM pruning methods.* We also tested the accuracy of pruning with ADMM as described in 5.1.2. As we have

changed the task into one-vs-all at the beginning before training for the last FC layer, we can only examine the accuracy of the network before and after the ADMM processes.

The result is: After trained for the FC layer, the accuracy is 99.85. And after the ADMM process, the accuracy is 99.83. What's more, the ADMM method pruned more than eighty percent for each layer.

But note the ADMM method involves the process of fine-tuning as a part of solving procedure, thus what it found is not exactly the "critical path" of the original network. What's more the "more than 80 percent" pruning rate is got by pruning the weights, but not the filters. As for the filters, the 80 percent pruning rate leads to more than 60 percent pruning rate in filter number.

6.4 Characteristics of Critical Path

6.4.1 *IOU of between differen critical paths.* We tested the path distribution of the different critical paths. For doing this, we calculated the IOU of the pruned critical paths. The result is shown in fig 4.

class	0	1	2	3	4	5	6	7	8	9
0	1	0.833561	0.830439	0.856354	0.836066	0.839206	0.840466	0.837321	0.844254	0.832936
1	0.833561	1	0.822991	0.848058	0.836066	0.847423	0.831687	0.842989	0.858279	0.828571
2	0.830439	0.822991	1	0.839206	0.82795	0.836066	0.831687	0.829816	0.832936	0.847423
3	0.856354	0.848058	0.839206	1	0.848693	0.873824	0.849329	0.838577	0.853793	0.836066
4	0.836066	0.836066	0.82795	0.848693	1	0.851877	0.842358	0.828571	0.836693	0.831063
5	0.839206	0.847423	0.836066	0.873824	0.851877	1	0.839836	0.837321	0.846788	0.834812
6	0.840466	0.831687	0.831687	0.849329	0.842358	0.839836	1	0.858279	0.846154	0.840466
7	0.837321	0.842989	0.829816	0.838577	0.828571	0.837321	0.858279	1	0.851877	0.826707
8	0.844254	0.858279	0.832936	0.853793	0.836693	0.846788	0.846154	0.851877	1	0.836066
9	0.832936	0.828571	0.847423	0.836066	0.831063	0.834812	0.840466	0.826707	0.836066	1

Fig. 4. **Intersection over union matrix of the pruned critical paths** Each number (x,y) represent the IOU value between the critical path of x and that of y

It can be found that the critical paths have large parts in common, which means the critical path we found may not be specific enough to different input classes. However, as often said in literature, the critical paths may share some same features in previous layers. Thus, we plotted out the trends of the IOU of different layers, as in fig 5.

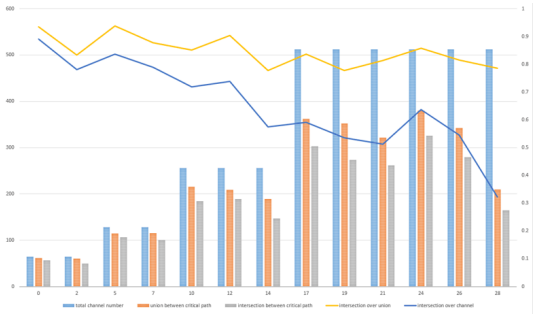

Fig. 5. **IOU trends of different layers** The x-axis represents the depth of the network, and the y-axis represents the IOU value. The yellow curve represents the IOU between the set of filters in critical path and that in the layer, The blue curve represents the average IOU between different class of inputs

From this figure, the discrepancy between the yellow line and the blue line shows the pruned filters mainly concentrates at those not useful for the whole e wtrk. No matter which input class might be, these filters are pruned. This observation shows that the algorithm for finding the critical paths is not very good at finding those response specifically to that class.

6.4.2 *Activation distribution.* To test the assumption that we can easily decide which the critical path might be at run-time, we have to look at the distribution of the activation. Below is a contrast between activation distribution on critical path and that on the network apart from critical path.

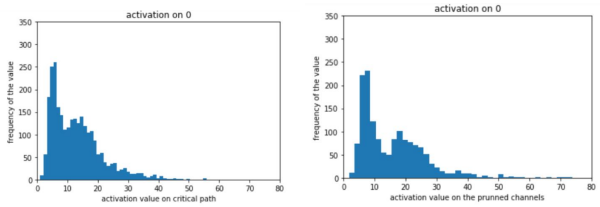

Fig. 6. **contrast between activation distribution on critical path and that on non-critical path** The x-axis is the activation average value in that feature map, the y-axis is the frequency

From the figures we can find that there are some difference when considering the activation distribution. Which might be useful for us in deciding the critical path, but the distribution is not that clear for us to make the decision easily.

7. CONCLUSION

Through hard work, we have learned and implemented some methods to find critical paths, and have done some experiments to explore critical paths and seek possible breakthroughs. As mentioned in the experiment section, we get a critical path in some sense through pruning. After that, we conducted researches as deep as we can on this extracted critical path, but did not find valuable features in it. This is indeed a bit frustrating. However, it is worthy of relief that this does not extinguish our proposal for the realization of the idea of network acceleration through critical paths, because we still have some methods to be practiced. Due to the limited time, we have not been able to dig out the features that are worth using in the critical path and use them for neural network prediction acceleration.However, through this project, we have a deeper understanding of the structure and nature of neural networks. The practical experience of TensorFlow will also be of great help to future life.

ACKNOWLEDGMENTS

We are very grateful to the teachers and assistants for their time and energy in this big assignment, best wishes!

REFERENCES

Zhuwei Qin, Fuxun Yu, Chenchen Liu, Liang Zhao, and Xiang Chen. 2018. Interpretable Convolutional Filter Pruning. *arXiv preprint arXiv:1810.07322* (2018).

Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. 2018. Interpret Neural Networks by Identifying Critical Data Routing Paths. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8906–8914.

Fuxun Yu, Zhuwei Qin, and Xiang Chen. 2018. Distilling Critical Paths in Convolutional Neural Networks. *arXiv preprint arXiv:1811.02643* (2018).

Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. 2018. Revisiting the Importance of Individual Units in CNNs via Ablation. *arXiv preprint arXiv:1806.02891* (2018).