Exploring Modularity of Neural Networks through Neuron Subselection

CMPUT 659: Project Report

Khaled Elbastawisy, Afia Afrin

1 Abstract

Understanding whether neural networks develop modular internal structures– where specific components specialize in distinct functions– remains a fundamental question in deep learning. In this project, we explore whether task-based modular structures emerge naturally in standard neural networks, and how they can be uncovered. We adopt a neuron-centric perspective in analyzing modularity where we identify functional subnetworks by selectively activating or suppressing individual neurons based on their behavior for a target task. Through a comparative analysis of two different neuron selection strategies, we examine the extent to which neural networks exhibit properties of specialization and reuse. While we agree with the baseline that modularity does not naturally emerge in neural networks, our conclusions regarding specialization and reusability diverge. We argue that the presence of these properties is highly sensitive to the experimental methodology, underscoring the need for a standardized definition of these properties and further research to determine whether neural networks consistently exhibit them.

2 Introduction

Neural networks (NNs) are widely considered to be black-box models with highly distributed internal representations. However, the possibility that these representations might be modular– that is, composed of distinct functional units– has attracted significant interest. Modular architectures can lead to improved interpretability, transferability, and robustness, especially in complex or multi-task learning settings.

In a recent study, Csordás et al. [1] explored whether functional modules naturally emerge in trained networks. They used binary weight masking to extract class-specific subnetworks and analyzed two properties: $P_{specialize}$, the tendency for different tasks to rely on different modules, and P_{reuse}, the reuse of the same module across instances of the same task. Their findings suggest that neural networks tend to specialize but do not reuse modules across tasks.

In this project, we revisit the question of modularity, but shift the focus from weights to neurons. Specifically, we investigate whether subnetworks composed of selected neurons can reveal the same modular properties. We propose and evaluate two neuron subselection strategies, (a) Neuron Thresholding: which selects neurons with the highest average activation for each class, and (b) Neuron Masking: which applies a Bayesian decision rule to force neurons ON, OFF, or leave them unchanged by masking.

We apply both techniques to CNNs trained on MNIST and CIFAR-10, and analyze their behavior under the $P_{specialize}$ and P_{reuse} metrics. Interestingly, the two approaches yield opposing results: neuron thresholding leads to high reuse but poor specialization, while neuron masking aligns with the baseline in showing strong specialization and weak reuse. These differences underscore how the choice of masking method shapes our conclusions about modularity in neural networks.

The implementation used in our experiments can be found in this github repository.

3 Background and Motivation

In [1], the authors investigate whether NNs naturally develop modular structures, where different parts of the network perform distinct, reusable functions. First, they define the idea of *functional modules* that is *'subsets of weights (i.e. subnetworks) responsible for performing a specific target functionality, such as solving a subtask of the original task'*. Then, they analyze functional modularity in pre-trained NNs using differentiable binary weight masks. For ease of reading, their methodology and experimental findings are described in a step-by-step manner below.

Extracting Modules: Modules are extracted by performing a weight-level introspection, revealing which parts of the network are responsible for specific tasks without retraining the entire model. First, they formulate a target task, that is a subtask of the original problem, corresponding to the specific function. Then, they train special masks that decide which weights (connections) are needed for that task. These masks are like on/off switches for each weight, and they are designed to be trainable while keeping the original network unchanged. The resulting mask then reveals the module (subnetwork) responsible for the target task.

Analyzing Usefulness of the Modules: Once a module is extracted for a particular subtask, the next question is how useful the module is? To analyze the fundamental properties of the modules, the authors introduce two metrics, namely $P_{specialize}$ and P_{reuse}. Here,

$P_{specialize}$: Does the network use different modules for different kinds of tasks?, and

P_{reuse}: Does the NN use the same module again when it needs to do the same task multiple times?

These two quantities are measured by considering the change in performance as a result of applying different masks corresponding to a target function.

3.1 Experimental Findings

They test modularity in multiple architectures, including Feedforward NNs (FNNs), CNNs, RNNs (e.g. LSTMs) and transformers using different datasets including synthetic tasks (e.g., addition/multiplication, double-addition) and real-world datasets (SCAN, CIFAR-10, and the Mathematics dataset). We discuss further regarding their experiments in Section 4. The key findings from their experiments are listed below:

- NNs often exhibit $P_{specialize}$ but lack P_{reuse}. To elaborate, their conclusion is Neural networks do specialize, as there is comparatively less weight-sharing in the hidden layers across different tasks. However, NNs do not reuse the same weights for repeating the same task.
- The authors apply their functional weight masking method to Convolutional Neural Networks (CNNs) trained on the CIFAR-10 image classification dataset. As they find, some weights are almost entirely responsible for specific classes, and not shared across others, which reinforces the lack of modular reuse. To elaborate, as shown in this work, removing weights for a particular class causes a big drop in performance on that class while the performance on other classes stayed mostly the same. This shows that some weights are almost exclusively used for certain classes and not shared across others. It shows a lack of modular reuse in CNNs: rather than building shared, reusable features, the network creates class-specific modules.
- Weight sharing in NN is often due to shared input/output structures, not because of functional similarity. In the addition/multiplication experiment, both operations used the same inputs and outputs, and the network showed some weight-sharing, even though these are very different functions. However, in the double-addition experiment, where the two additions had separate input/output units (i.e., different I/O), the network had a lower weight-sharing ratio, even though it was doing the same operation twice.
- On algorithmic tasks such as SCAN or mathematical reasoning problems, neural networks often fail to learn generalizable, compositional solutions. Instead of developing reusable building blocks, they rely on combination-specific weights, effectively learning each case in isolation– even when the same underlying logic applies. Take the SCAN dataset as an example: it requires a model to map natural language commands (e.g., "jump twice") to action sequences (e.g., "JUMP JUMP"). The task is specifically designed to test whether models can

Algorithm 1 Get Subnetworks using the Neuron Thresholding Approach

```
1: function GetSubnetworks($m$, $percentile$, $c$)
2:     $activation\_model$ ← intermediate model returning layer activations
3:     $X_c$ ← data samples from class $c$
4:     $A$ ← $activation\_model$.predict( $X_c$)                    ▷ get important neurons for class $c$
5:     for each layer $l$ in $model$ do
6:         if $l$ is a Conv2D layer then
7:             $avg\_activation$ ← mean of $A$ across spatial and batch dimensions
8:         else if $l$ is a Dense layer then
9:             $avg\_activation$ ← mean of $A$ across batch
10:        end if
11:        $threshold$ ← percentile($avg\_activation$, $percentile$)
12:        Mark neurons that have $avg\_activation > threshold$ as important
13:    end for
                                           ▷ Use the important neurons to create a subnetwork
14:    Build a neural network, $N\_s_c$, consisting only of the important neurons and their associated weights
15:    Return the subnetwork $N\_s_c$
16: end function
```

systematically generalize– such as applying the known rule "twice" to an unseen verb like "play" or "drink". As shown in this work, typical neural networks do not generalize in this way. Rather than treating "twice" as a reusable adverb that can be applied to any verb, the network learns distinct weights for each seen combination– such as "run twice" or "walk twice"– memorizing them as wholes. This behavior indicates a failure to capture the compositional structure of the task and suggests a lack of true modularity in the learned representations.

Our project is based on this work but takes a different methodological route. We hypothesize that weight-masking might not the best way to find modularity in NNS, since by changing the weights of the model, we can essentially "retrain" the neural network to attain any function we like. Therefore, we aim to extract modules in neural networks by subselecting neurons. In the following sections, we describe the methodology and experimental setup used to test this hypothesis and present the corresponding results.

4 Methodology

We use two CNN models trained on MNIST and CIFAR-10 respectively for all of our experiments. We test two different approaches to subselect neurons of the CNN models. This section describes the two approaches, namely *Neuron Thresholding* and *Neuron Masking*.

4.1 Neuron Thresholding

The underlying hypothesis of this idea is simple: if a neural network exhibits inherent modularity, it should function in a modular fashion. For a classification task, this would imply that the network allocates distinct sets of neurons to handle different classes, with minimal overlap. In this experiment, we set out to test this idea by identifying class-specific subnetworks.

For any target class c, we begin by passing all samples belonging to class c through a pre-trained convolutional neural network (CNN) and record the activation values of neurons across each of the Dense and Conv2D layers[1]. For every layer, We compute the average activation of each neuron and identify those whose activations lie above the 70^th percentile of their respective layer. These neurons are designated as *important neurons* for class c.

To construct a class-specific subnetwork, we retain only the important neurons for that class and their corresponding weights. Specifically, we apply a binary mask that sets the weights and biases associated with unimportant

[1]We exclude layers such as Flatten, Dropout, BatchNormalization, and MaxPooling since they do not contain learnable parameters or neuron activations in the conventional sense. Rather, we focus exclusively on layers that apply transformations involving neuron activations and contribute directly to feature learning.

Algorithm 2 Get Subnetworks using the Neuron Masking Approach

```
function GetSubnetwork($model$, $c$, $t_{\text{high}}$, $t_{\text{low}}$, $\tau$, $f_{\text{layer}}$)
    $activation\_model$ ← intermediate model returning all layer activations
    $X_c$ ← data samples from class $c$
    $A$ ← $activation\_model$.predict( $X_c$)
    for each weighted layer $l$ in $model$ do
        $a_l$ ← mean of $A$ across batch (and spatial dimension if $l$ is Conv2D)
        Compute $(\mu_j, \text{SE}_j)$ for each neuron $j$ in layer $l$
        $P_{\text{high}}[j] \leftarrow 1 - \Phi\left(\frac{t_{\text{high}} - \mu_j}{\text{SE}_j}\right)$
        $P_{\text{low}}[j] \leftarrow \Phi\left(\frac{t_{\text{low}} - \mu_j}{\text{SE}_j}\right)$
        for each neuron $j$ in $l$ do
            if $P_{\text{high}}[j] > \tau$ then
                $mask_l[j] \leftarrow 1$
            else if $P_{\text{low}}[j] > \tau$ then
                $mask_l[j] \leftarrow 0$
            else
                $mask_l[j] \leftarrow -1$
            end if
        end for
    end for
    return BuildMaskedNetwork($model$, $mask$)
end function
```

neurons to zero, effectively deactivating them. This results in a pruned version of the original network that is specialized for class c, preserving only the most relevant features for that class. The complete procedure is described in Algorithm 1.

4.2 Neuron Masking

We extend the approach presented in [2] to create masks for each neuron. Given a pre-trained CNN, we collect neuron activations for target class c by passing all class-c samples through the model. For every neuron in each weighted layer (i.e. Conv2D or Dense), we compute the mean and standard error of its activation values across those samples. If a neuron is from a convolutional layer, activations are averaged over both spatial dimensions and samples to yield a per-channel value. For dense layers, activations are averaged over the batch.

We assume that each neuron's true mean activation over class-c inputs follows a normal distribution and compute the probability that this mean lies above a high threshold t_{high} or below a low threshold t_{low}. These probabilities are given by:

$$P_{\text{high}} = 1 - \Phi(\frac{t_{\text{high}} - \mu}{\text{SE}}), \quad P_{\text{low}} = \Phi(\frac{t_{\text{low}} - \mu}{\text{SE}})$$

where Φ is the cumulative distribution function of the standard normal distribution, μ is the sample mean, and SE is the standard error. These probabilities reflect how confident we are that the neuron is consistently active or consistently inactive for class c. Based on the computed probabilities, we assign a mask value to each neuron:

mask = 1: If $P_{\text{high}} > \tau$, the neuron is forced **ON** by zeroing its weights and setting its bias to 1.

mask = 0: If $P_{\text{low}} > \tau$, the neuron is forced **OFF** by zeroing its weights and setting its bias to 0.

mask = −1: If neither condition holds, the neuron is left unchanged and allowed to compute normally.

Here, τ is the minimum posterior confidence required to make a neuron-masking decision. It is used to threshold the probabilities P_{high} and P_{low} during Bayesian inference. [2]

[2]Higher τ = more conservative masking; lower τ = more aggressive pruning. In our experiment, we use $\tau = 0.8$.

This approach allows us to retain parts of the original network's computation graph, while enforcing strong priors on the behavior of confident neurons. Algorithm 2 shows the pseudocode for this approach. Note, the $BuildMaskedNetwork$ function in Algorithm 2 simply builds a subnetwork by modifying the original network in the following way–

Neurons with $mask = 1$ are forced to always output 1 by zeroing incoming weights and setting the bias to 1.

Neurons with $mask = 0$ are zeroed and biased to always output 0.

Neurons with $mask = -1$ are left untouched and allowed to compute based on incoming activations.

Temperature Scaling for Confidence Calibration: Due to pruning and forced activations, the output logits of masked subnetworks may be overconfident or miscalibrated. To mitigate this, we apply temperature scaling– a post-processing technique applied to a model's logits (raw outputs before softmax) to improve calibration. During model inference stage, the logits are divided by a learned temperature parameter $T > 1$, producing softened softmax outputs:

$$\hat{y}^i = \frac{\exp(z^i/T)}{\sum_j \exp(z^j/T)}$$

5 Experimental Analysis

We perform two separate experiments. First, we investigate the first key finding from the baseline work (refer to Section 3.1 for the key findings) using the two neuron subselection approaches described in the previous section. Based on the insights gained from this experiment, we proceed with a second experiment focused on analyzing specialization in CNNs, specifically using the *Neuron Masking* method for subselecting neurons. Further details behind this decision are provided shortly as we walk through the experimental setup and the corresponding results.

5.1 (Experiment 1) Analyzing Fundamental Properties of Modules

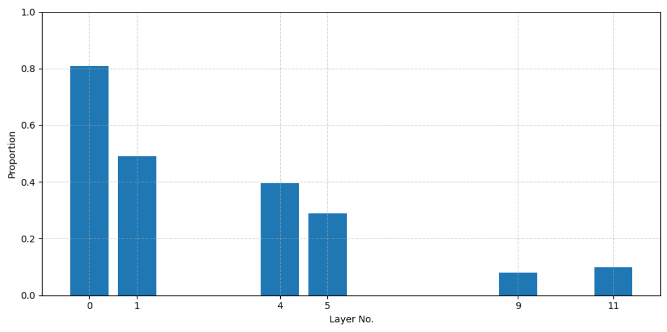
To evaluate the $P_{specialize}$ property, we examine the extent to which active neurons are shared across different subtasks. Specifically, we randomly select two distinct classes, c_1 and c_2, and define the subtasks as follows: Subtask 1 involves identifying instances of class c_1, while Subtask 2 involves identifying instances of class c_2. For each subtask, we subselect active neurons from each of the Dense and Conv2D layers using the neuron subselection approaches outlined in Section 4. We then compute the proportion of active neurons shared between the two subtasks for each layer (i.e. the number of active neurons per layer that are common for the two subtasks divided by the total number of neurons in that layer). For this experiment, less number of shared neurons indicate the presence of $P_{specialize}$ property in the network. This process is repeated ten times with different pairs of random classes, and the average sharing ratio of neurons per layer is reported in Figure 3.

To investigate the P_{reuse} property, we perform a similar experiment. A single class c is randomly selected, and its instances are split into two groups. The subtasks are then defined as follows: Subtask 1 consists of identifying instances from group 1, and Subtask 2 consists of identifying instances from group 2. Since both groups originate from the same class, a high degree of neuron sharing across all weighted layers would indicate the presence of the P_{reuse} property in the original network. The results are reported in Figure 4. We discuss the results in the following subsection.

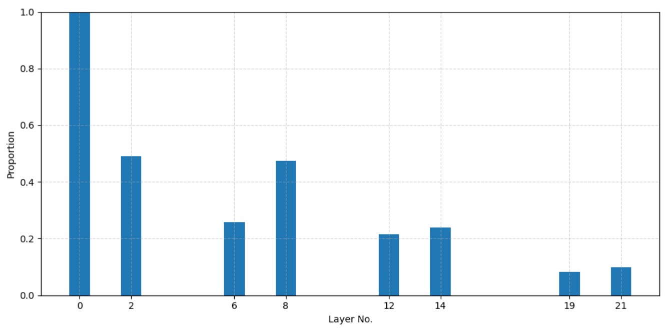
5.1.1 Results

Results for this experiment obtained from the two neuron subselecting techniques, namely Neuron Thresholding and Neuron Masking, are described below.

Neuron Thresholding: As shown in Figure 1, neither of the two networks demonstrates the $P_{specialize}$ property. Shared neurons are present in every layer, and in some cases, the proportion of shared neurons exceeds 80%, indicating a lack of strong specialization. Regarding the P_{reuse} property, the outcome is quite opposite. Figure 2 highlights a consistently high degree of neuron sharing across nearly all layers. This pattern suggests a significant level of reuse,

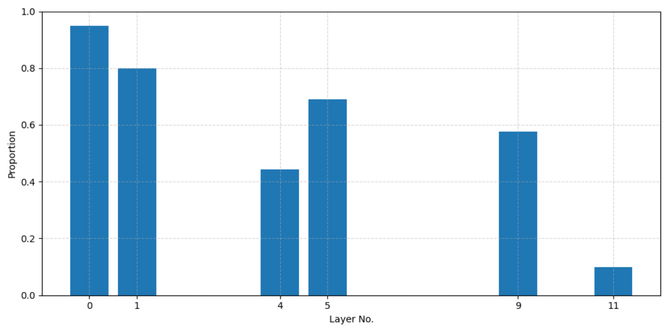


(a) $P_{specialize}$ with MNIST dataset

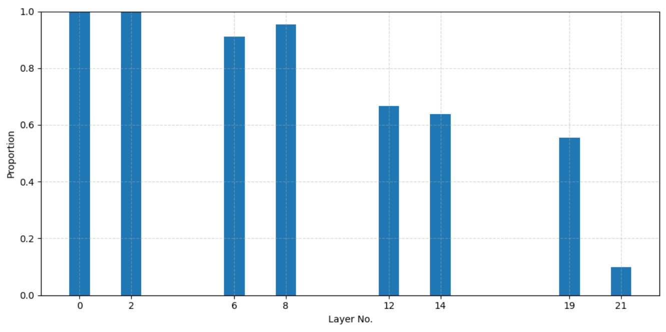


(b) $P_{specialize}$ with CIFAR-10 dataset

Figure 1: Proportion of active neurons shared between modules across different subtasks using *Neuron Thresholding*. Computed and plotted for each Dense or Conv2D layer only.



(a) P_{reuse} with MNIST dataset



(b) P_{reuse} with CIFAR-10 dataset

Figure 2: Proportion of active neurons shared between modules across similar subtasks using *Neuron Thresholding*. Computed and plotted for each Dense or Conv2D layer only.

implying that the network learns features that are broadly applicable across the similar subtasks involved in the P_{reuse} setting.

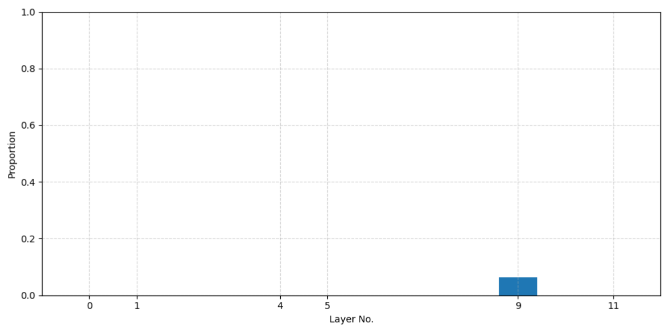
Interestingly, the findings of this experiment contradict the baseline results reported in [1]. To elaborate, we found that P_{reuse} exists in networks while $P_{specialize}$ does not but the baseline work found the opposite. We discuss more about this contradictory findings later in Section 6.

Neuron Masking: Now we present results for the same experiment with a different neuron subselection technique[3] that creates masks for selecting active neurons. As shown in Figure 3, the network trained on MNIST exhibits a clear $P_{specialize}$ property, with minimal neuron overlap observed only in layer 9. The network trained on CIFAR-10 presents a slightly different pattern with more amount of shared neurons (i.e. 7 out of 8 layers have shared neuorns). However, the proportion of shared neurons remains below 20% in all of the layers. These findings suggest that $P_{specialize}$ is still present in the CIFAR-10 model– albeit to a lesser extent than in the MNIST model. This result aligns well with the observations reported in the baseline study [1].

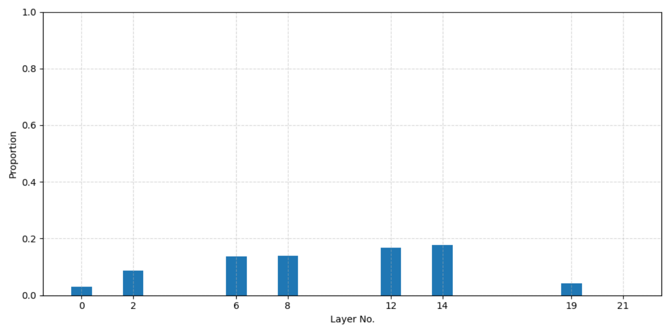
As shown in Figure 4, both networks demonstrate limited reusability. Even for closely related tasks on the MNIST dataset– such as identifying instances from the same class– the proportion of shared neurons is minimal, with overlap occurring in only a single layer and remaining below 30%. The distribution of shared neurons in the CIFAR-10 model closely aligns with that of the $P_{specialize}$ experiment, with a few layers (e.g., 12, 14, and 19) showing slightly, but not significantly, higher neuron sharing, further indicating low reusability. These results suggest that the P_{reuse} property is largely absent in both networks, consistent with the findings reported in the baseline study [1].

Note the interesting shift in outcomes when we change the neuron subselection technique. While the previous experiment yielded results that contradicted the baseline, the current approach produces findings that are consistent

[3]Detailed description of this neuron subselection technique is in Section 4.2

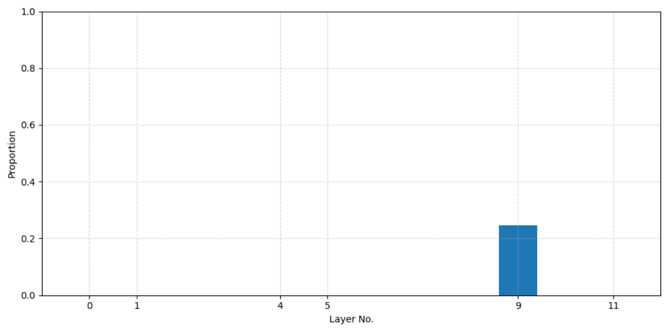


(a) $P_{specialize}$ with MNIST dataset

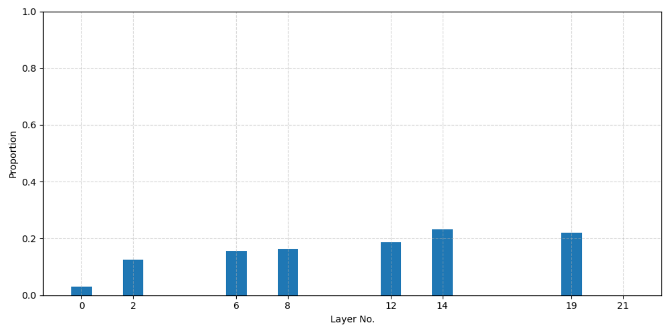


(b) $P_{specialize}$ with CIFAR-10 dataset

Figure 3: Proportion of active neurons shared between modules across different subtasks using *Neuron Masking*. Computed and plotted for each Dense or Conv2D layer only.



(a) P_{reuse} with MNIST dataset



(b) P_{reuse} with CIFAR-10 dataset

Figure 4: Proportion of active neurons shared between modules across similar subtasks using *Neuron Masking*. Computed and plotted for each Dense or Conv2D layer only.

with it. We will address this issue later in the discussion section– let us first wrap up all the experiments!

5.2 (Experiment 2) Analyzing Specialization in CNNs

Based on the findings of the previous experiment, the *Neuron Thresholding* approach for subselecting neurons failed to extract specialized modules from the network. Therefore, in this experiment where we analyze the specialization in CNNs, we naturally discard reporting results for this subselection approach. Rather, we analyze the specialization of CNN models in light of the *Neuron Masking* approach.

As discussed in the previous section, while using the *Neuron Masking* approach, we found that NNs show some degree of specialization i.e. it has class-specific modules. If this is true, we should be able to isolate and remove parts of the network that are only needed for one class, without hurting performance on others. Similar to the baseline work, we use ablation study to test this hypothesis and present the results in this section.

We begin by evaluating the original model on the test set and recording its confusion matrix. Then, for a given class c, we create an ablated version of the model by removing the neurons that are active for that class– this is done by zeroing out all the outgoing weights, as well as the bias terms, for those neurons. We evaluate this ablated model on the same test set to obtain a new confusion matrix. Finally, we compute the difference between the original and ablated confusion matrices for each class, and present the results for MNIST in Figure 5 and for CIFAR-10 in Figure 6. Results corresponding to this experiment are discussed below.

5.2.1 Results

As shown in Figure 5, removing the active neurons associated with any class c from the MNIST model leads to a significant drop in accuracy for that specific class, as reflected by the negative value in the diagonal cell corresponding to class c in the confusion matrix. In contrast, the true positive and false negative rates for the remaining classes are largely unaffected– all other cells of the confusion matrix is mostly stable except some cells incurring some false positives (i.e. the dark red cells). This increase in false positives for the other classes is expected though, as samples from the removed class are now misclassified as belonging to those classes. This clearly proves the high specialization in the MNIST model, as we observed in Figure 3 and discussed the previous section.

The results for the CIFAR-10 dataset, shown in Figure 6, reveal a more nuanced pattern. While there is evidence of class-specific specialization– indicated by the strong drop in the diagonal entry for the removed class– we also observe a broader decline in accuracy across all classes, as reflected by the overall bluish tone along the diagonal. This aligns with our earlier observation (see Figure 3 for reference) that the CIFAR-10 model exhibits a higher degree of neuron sharing compared to the MNIST model. As a result, it shows less pronounced specialization, which is once again evident in this experiment.

6 Discussion

The baseline work by Csordás et al. [1] presents two key conclusions:

- Modularity does not naturally emerge in neural networks unless it is explicitly enforced.
- Neural networks often exhibit the $P_{specialize}$ property but typically lack P_{reuse}.

Our experimental findings support the first conclusion– without architectural or training constraints, modularity does not arise spontaneously. However, our results offer a nuanced perspective on the second point. We found that the way in which modules are constructed has a significant influence on the observation of whether networks appear to exhibit specialization or reusability. We used two different approaches for constructing modules and got two different results. When modules are defined as the top n^{th} percentile of neurons based on their activation values for a given class c, we observe a high degree of neuron sharing across tasks. According to the baseline's definition, these modules are "reusable." However, with high degree of shared neurons across different subtasks, they lack specialization. Conversely, when we create modules using neuron masks, as described in Section 5.2.1, we observe the opposite trend. These modules demonstrate better specialization with few amount of shared neurons, but show limited reusability across similar tasks.

From these observations, we conclude that: When modules (subnetworks) are used to analyze a neural network's reusability or specialization, the conclusions drawn are highly influenced by the way in which these modules are extracted from the original model. This calls for further research to develop standardized, objective approaches for defining and identifying modularity in neural networks.

7 Concluding Remarks

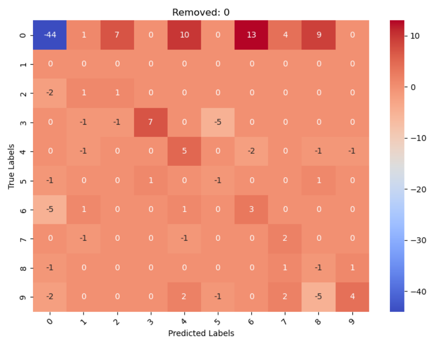
Can we extract meaningful modules from a neural network using neuron-level masking? Our results suggest that the answer is likely no– or at least, not in the way we might initially hope. But there is an important caveat: the outcome heavily depends on how we define a "module" and what we expect it to achieve. If we isolate a module/ subnetwork based only on its performance on the targeted function and expect it to perform well both on a target task and actively avoid supporting unrelated tasks, that may be too ambitious– especially without explicit structural constraints. Task-based modularity, while intuitive, may place overly rigid expectations on how neural representations are organized. Instead, we propose shifting focus toward pattern-based modularity: rather than associating modules with high-level tasks or classes, we should explore whether modules consistently activate in response to specific input patterns or features. For example, can we find a set of neurons that reliably activate whenever a particular visual pattern, such as a vertical edge or a circular shape, appears, regardless of the class label? This perspective aligns more closely with how biological systems are thought to operate and may lead to more robust, interpretable, and compositional representations in deep networks.

Another research direction can be introducing a shift in the focus i.e. instead of looking for modular subnetworks, we can aim for designing modular neural networks. Much like the approach taken in physics-informed neural networks where physical laws are directly incorporated into the model during its training process, we could impose architectural or learning constraints that encourage the emergence of modular structures– enabling networks not only to achieve high performance but also to develop interpretable and reusable functional components.

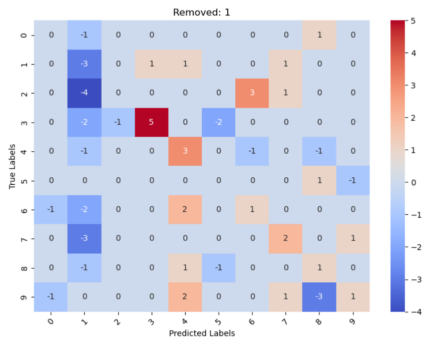
References

[1] R. Csordás, S. van Steenkiste, and J. Schmidhuber, “Are neural nets modular? inspecting functional modularity through differentiable weight masks,” *arXiv preprint arXiv:2010.02066*, 2020.

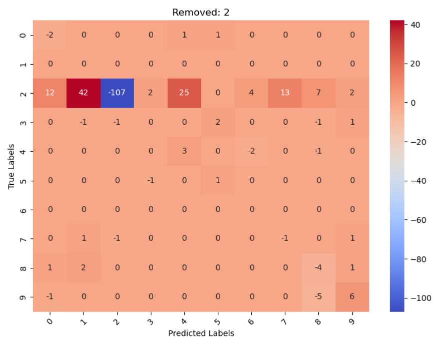
[2] M. Alikhasi and L. H. Lelis, “Unveiling options with neural decomposition,” *arXiv preprint arXiv:2410.11262*, 2024.



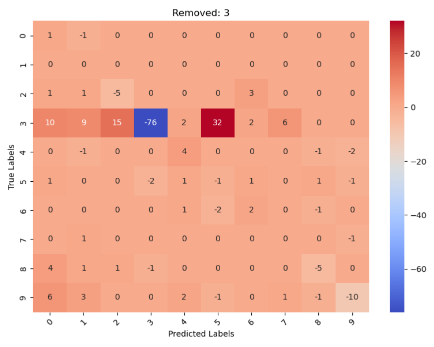
(a)



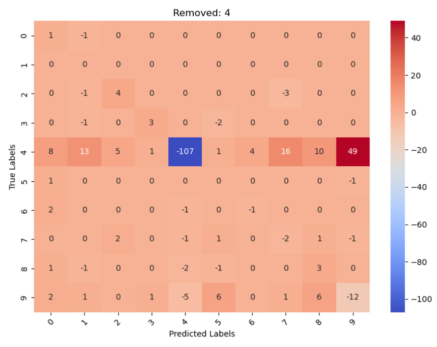
(b)



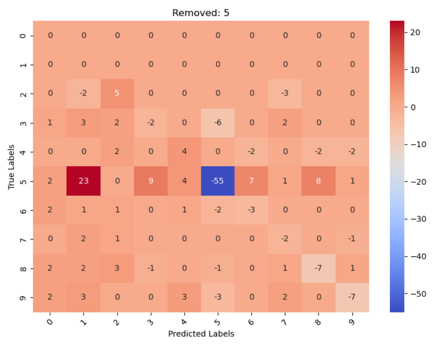
(c)



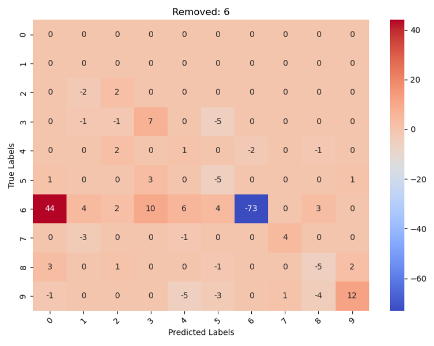
(d)



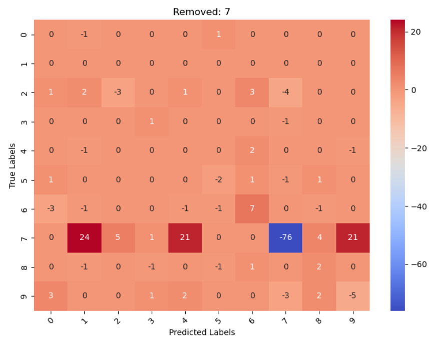
(e)



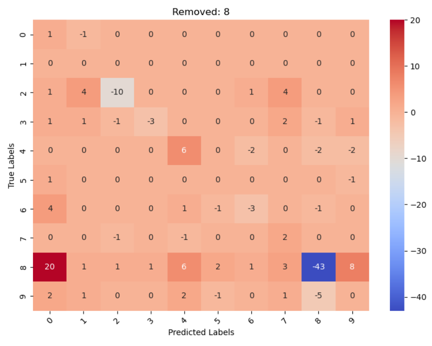
(f)



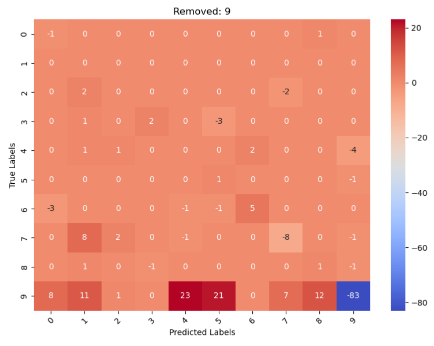
(g)



(h)

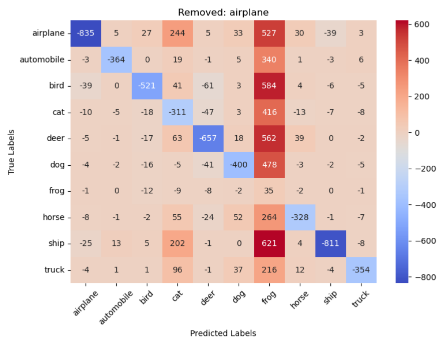


(i)

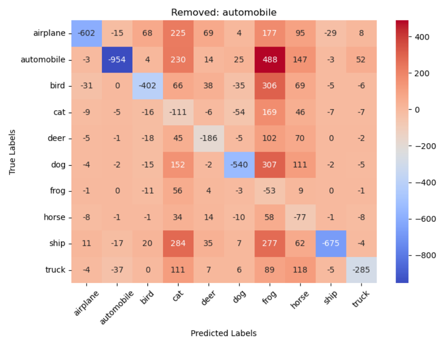


(j)

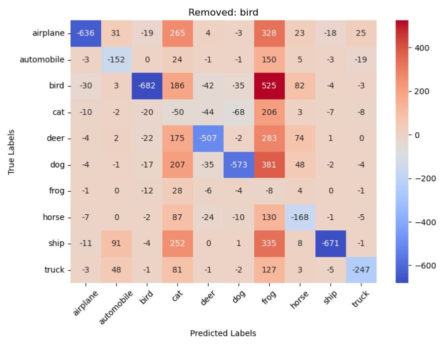
Figure 5: The change in confusion matrix for all MNIST classes, when class indicated by the caption, is removed. Note: negative values indicates less samples are being assigned to that class and vice-versa.



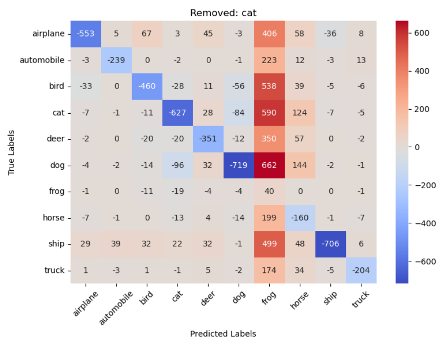
(a)



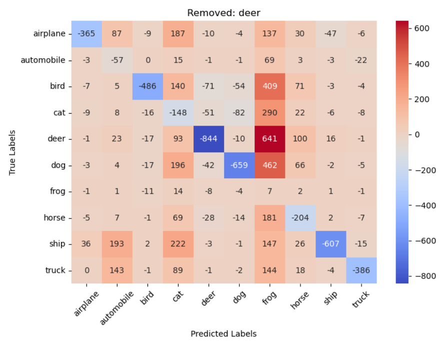
(b)



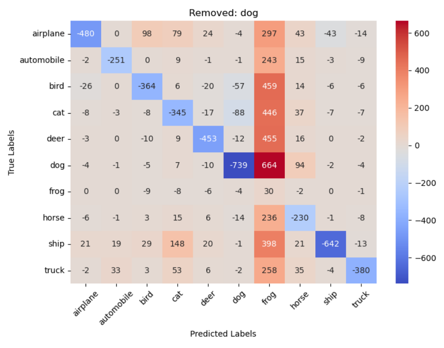
(c)



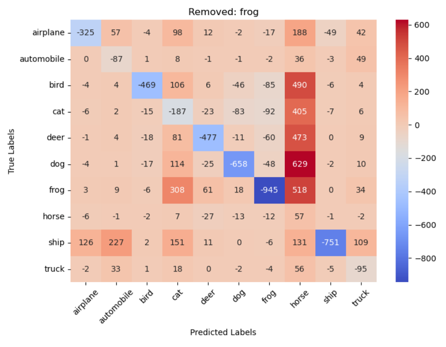
(d)



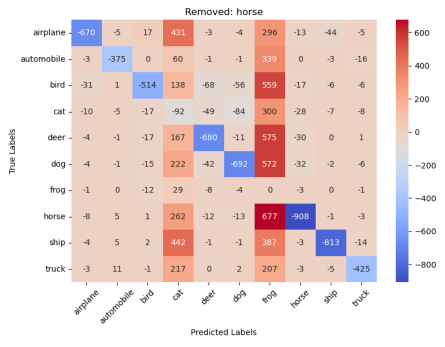
(e)



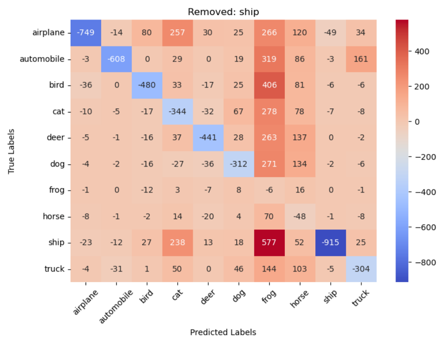
(f)



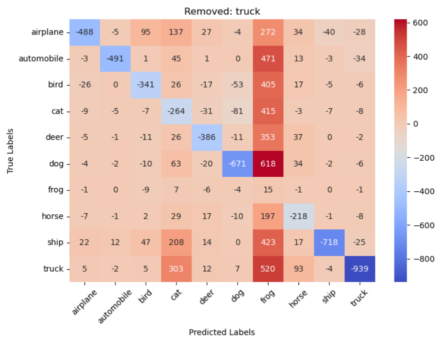
(g)



(h)



(i)



(j)

Figure 6: The change in confusion matrix for all CIFAR-10 classes, when class indicated by the caption, is removed. Note: negative values indicates less samples are being assigned to that class and vice-versa.