**Hardware Accelerators for Machine Learning**

**236509**

**Spring 2023**

**Adam Ghadban: 322661802**

# Abstract

In the age of widespread adoption of machine learning algorithms, functional safety (FuSa) becomes increasingly important, particularly in applications such as autonomous cars and Internet of Things (IoT) devices where model accuracy can directly impact user safety. This project focuses on the potential degradation of Convolutional Neural Network (CNN) and networks based on CNN’s (ResNet18), performance due to fault injections into the datapath of systolic arrays, the hardware accelerators commonly used in deep learning tasks. Using a CNN model trained on the CIFAR-10 dataset as our case study, we explored the classification error introduced by various fault models. This work builds upon the study conducted by Kundu et al. on Multi-Layer Perceptrons (MLPs) and extends the investigation to CNNs, thus providing a more comprehensive understanding of fault impacts in deep learning hardware accelerators.

# Project Initiation and Decision Making

At the outset of this project, our central challenge was to devise an effective strategy to simulate a systolic array. We contemplated several approaches, each with its own strengths and considerations:

1. The first approach considered was to inject faults at a high-level into a PyTorch model. This method would require us to presuppose that each fault injected into the model could be interpreted as a specific fault in a systolic array.
2. The second idea was to translate the `conv2d` operation in PyTorch into a hardcoded matrix multiplication (`matmul`). This translation would make it easier to correlate faults in the PyTorch model with potential faults in a systolic array.
3. Lastly, we contemplated the use of a systolic array simulator, which could either be a software-based or hardware-based simulation. This approach would provide a more realistic representation of a systolic array but would likely be more complex and time-consuming to implement.

After careful deliberation, given the project's limited timeline, we decided to proceed with the first approach. We concluded that injecting faults at a high level into a PyTorch model would provide a balance of ease of implementation and sufficient fidelity for our initial exploration of fault impacts on a Convolutional Neural Network deployed on a systolic array. This decision allowed us to quickly move forward with fault model creation, fault injection, and subsequent analysis.

# Project Workflow

Our research journey began with the training of a PyTorch ResNet18 model on the CIFAR-10 dataset, achieving a test set accuracy of 86%. However, understanding the need for versatility in model selection, we enabled the usage of various network models from RobustBench, a comprehensive repository of robust models, to allow researchers the flexibility to explore beyond the ResNet architecture.

Following this, we developed a highly adaptable generic fault model. This fault model was designed to provide a comprehensive simulation of a wide array of fault scenarios. It was capable of parsing layer names in the given model, targeting a specific layer, selecting the number of faults, determining the target of the fault (weight, bias, or memory), and deciding on the number of bits to be flipped. The fault model also enabled control over the distribution of bit flips (uniform or Gaussian) and the distribution of faults across the target. This level of customization was vital in creating a wide-ranging fault model capable of reflecting real-world adversarial attacks or spontaneous system corruptions.

To facilitate the fault injection process, we developed an injector that could introduce the designated faults into the chosen model. This allowed us to measure the impact of the faults on the accuracy of the model.

In our endeavor to conduct an exhaustive investigation, we aimed to establish correlations between multiple fault parameters in the model (such as the number of faults, distribution of bit flips, distribution of faults, etc.) and the resultant model accuracy. This was crucial in understanding the direct and indirect impacts of various fault factors on the accuracy of a CNN deployed on a systolic array.

To conduct the testing, we used a subset of 50 images from the CIFAR-10 dataset. Considering the computational demands of the fault injection process and subsequent model cleanup, we chose to inject faults individually. This helped us in calculating the average impact of each fault injection on model accuracy.

To maximize computational efficiency and performance, we exploited parallel computing. We ran up to three different fault models concurrently on separate GPUs. This approach allowed us to explore a wider range of fault impacts in a reduced timeframe.

Importantly, we provided users the flexibility to select the fault model via the Command Line Interface (CLI), further enhancing the adaptability and user-friendliness of our experimental setup.

This versatile and comprehensive methodology enabled us to gain deep insights into the effects of various fault parameters on the performance of CNNs deployed on systolic arrays. It also laid a strong foundation for future explorations into effective mitigation strategies, thus taking us a step closer to ensuring functional safety in machine learning applications.

# Data Aggregation

Upon completion of the fault injection process, we recorded the impact on accuracy for each individual fault injection. Considering that each fault was cleaned before injecting the next, we ensured that the impact assessment of each fault was isolated from others. The accuracy measurements corresponding to each fault injection were then compiled into a JSON file for further evaluation.

To visually interpret the data and comprehend the correlations between fault parameters and model accuracy, we constructed a script to parse the results encapsulated in the JSON file and create illustrative graphs. Each graph displayed the degradation in model accuracy in relation to max, min and avg of accuracy degradation.

In this case, the x-axis represented the specific fault model number (e.g., Fault Model 1, Fault Model 2, etc.), and the y-axis displayed the associated drop in accuracy. This representation was instrumental in illustrating the unique impacts of individual fault models on model accuracy.

Recognizing the significance of the number of faults and the number of bits flipped in a fault model, we categorized all graphs based on these two parameters. This classification allowed us to detect patterns, thereby identifying which factors exerted the most considerable influence on accuracy degradation.

Through this systematic and visual approach to data aggregation and analysis, we effectively gleaned comprehensive insights into the relationships between different fault parameters and model accuracy, thereby significantly enhancing our understanding of the effects of faults on Convolutional Neural Networks deployed on systolic arrays.

# Distinguishing Aspects of Our Study

While our study draws on the methodological foundations established in the referenced paper, there are several critical distinctions that differentiate our work. Here are the key differentiating factors:

# **Network Architecture**: Unlike the paper that focused on Multilayer Perceptrons (MLPs), our study investigates Convolutional Neural Networks (CNNs). This shift in focus allows us to explore the more complex and spatially-aware structure of CNNs, which is more pertinent to many modern applications such as image classification.

# **Fault Types**: In contrast to the original paper, we did not simulate stuck-at faults. This is due to the hard translation of a stuck-at fault in a PE to a high-level fault.

# **Bit Flip Granularity**: We considered lower bit flips in the datapath, exploring the implications of faults at a finer level of detail.

# **Memory Faults**: Our study also accounted for faults in memory, specifically in the stored image data. This allowed us to observe how data integrity issues could affect the performance of CNNs.

# **Fault Distribution**: Another unique aspect of our research was the consideration of fault distribution within the target. This added another layer of complexity to our fault models, allowing us to investigate how the spatial distribution of faults could affect accuracy.

# **Bit Flip Strategy**: Lastly, we implemented a non-strict approach to bit flips. This meant that we allowed for multiple adjacent bits to be flipped together, thus introducing the possibility of more complex and significant faults.

# These distinguishing factors not only set our work apart from the original paper, but also allowed us to explore new dimensions in the study of fault impacts on the performance of systolic array-based deep learning hardware accelerators. This broader and more intricate exploration offers insights that can help in designing more resilient and reliable systems.

# Data Analysis

Note that the network was not quantized and was trained in float32. Each fault model was run 100 times, and the result we see in the graphs are the average. This was done to get a more general picture of the relation between the fault model and accuracy,

## Bit granularity and its relation to FuSa violations

**Fault model to lower bit flips:**

Fault\_target = bias/memory/weight

Number\_of\_faults = 20

Num\_bits\_to\_flip = 6

Bits\_to\_flip = 1,2,3,4,5,6

Fault\_distribution = gaussian/uniform

Target\_layer = {

Bias = {

BN layer at the beginning of the NN, BN at the early middle, BN at the end of middle, BN at the end

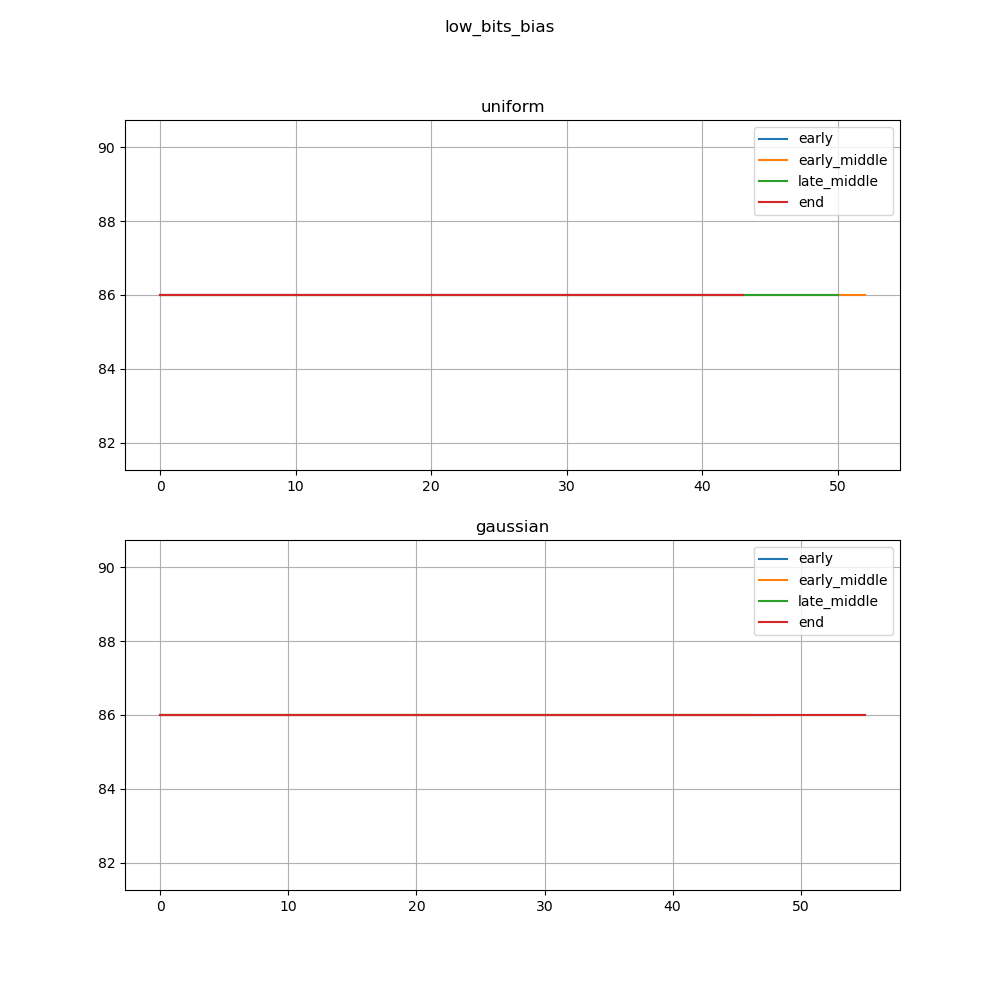
},

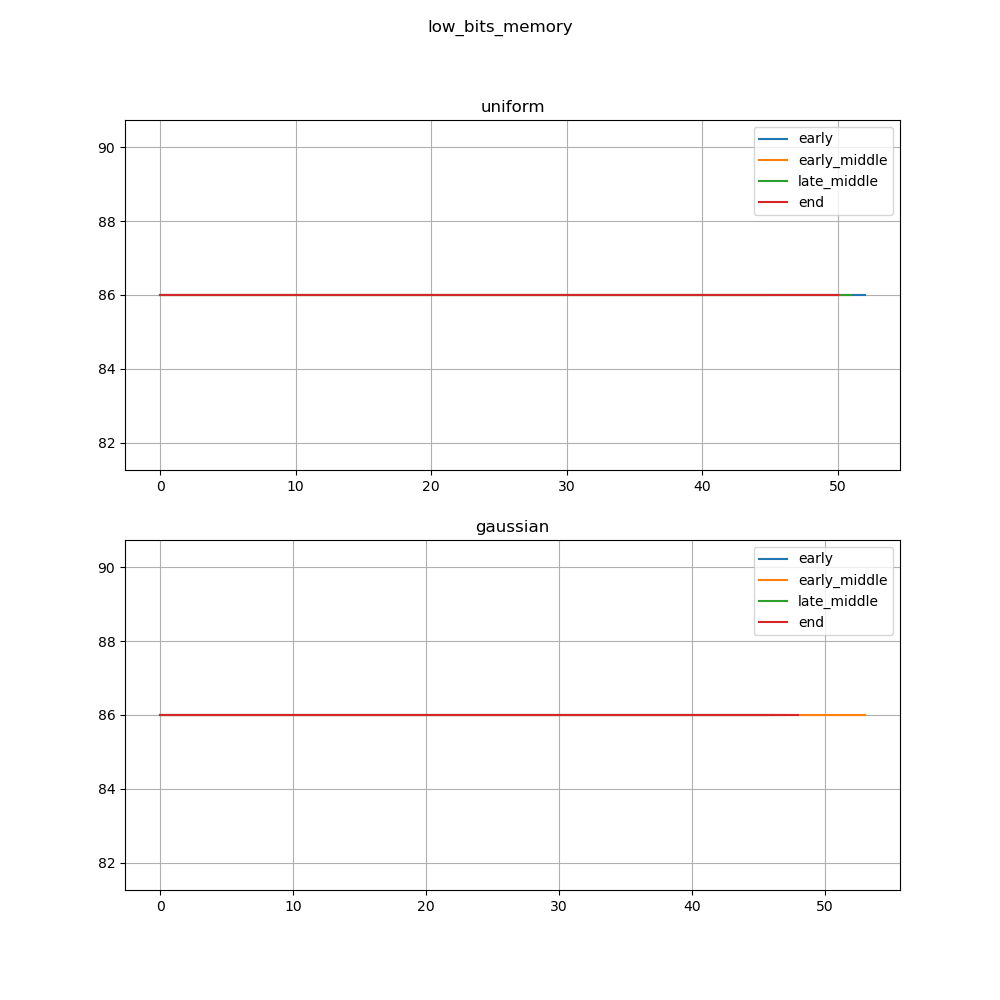
Memory, weights = {

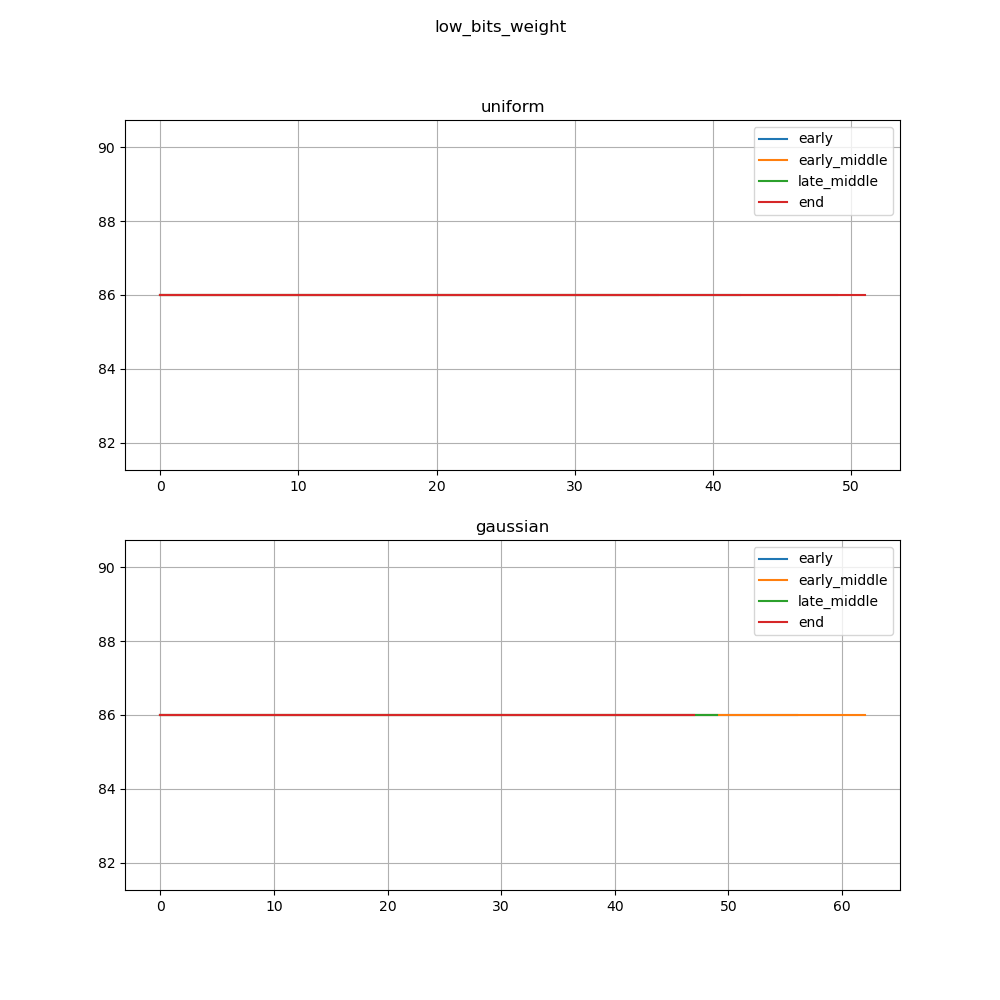
Same but conv layers instead of BNs.

}

}







We can see that low bits flips (1, 2, 3, 4, 5 and 6) bits did not affect the accuracy of the model.

This can be attributed to the fact that the model is in float32, were the LSB’s flips does not affect the number value as much as flipping in int32. (Mantissa flips).

**Fault model to middle bit-flips:**

Fault\_target = bias/memory/weight

Number\_of\_faults = 20

Num\_bits\_to\_flip = 5

Bits\_to\_flip = 10, 11, 12, 13, 14

Fault\_distribution = gaussian/uniform

Target\_layer = {

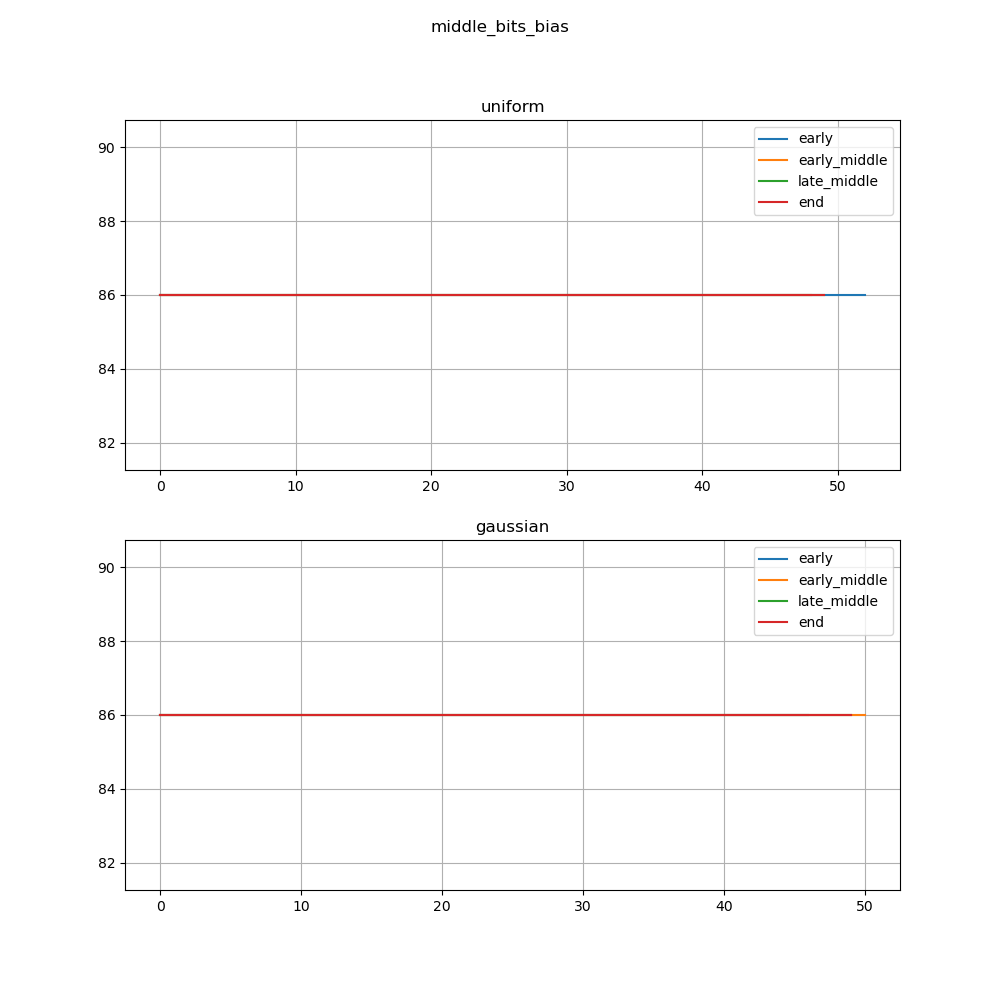
Bias = {

BN layer at the beginning of the NN, BN at the early middle, BN at the end of middle, BN at the end

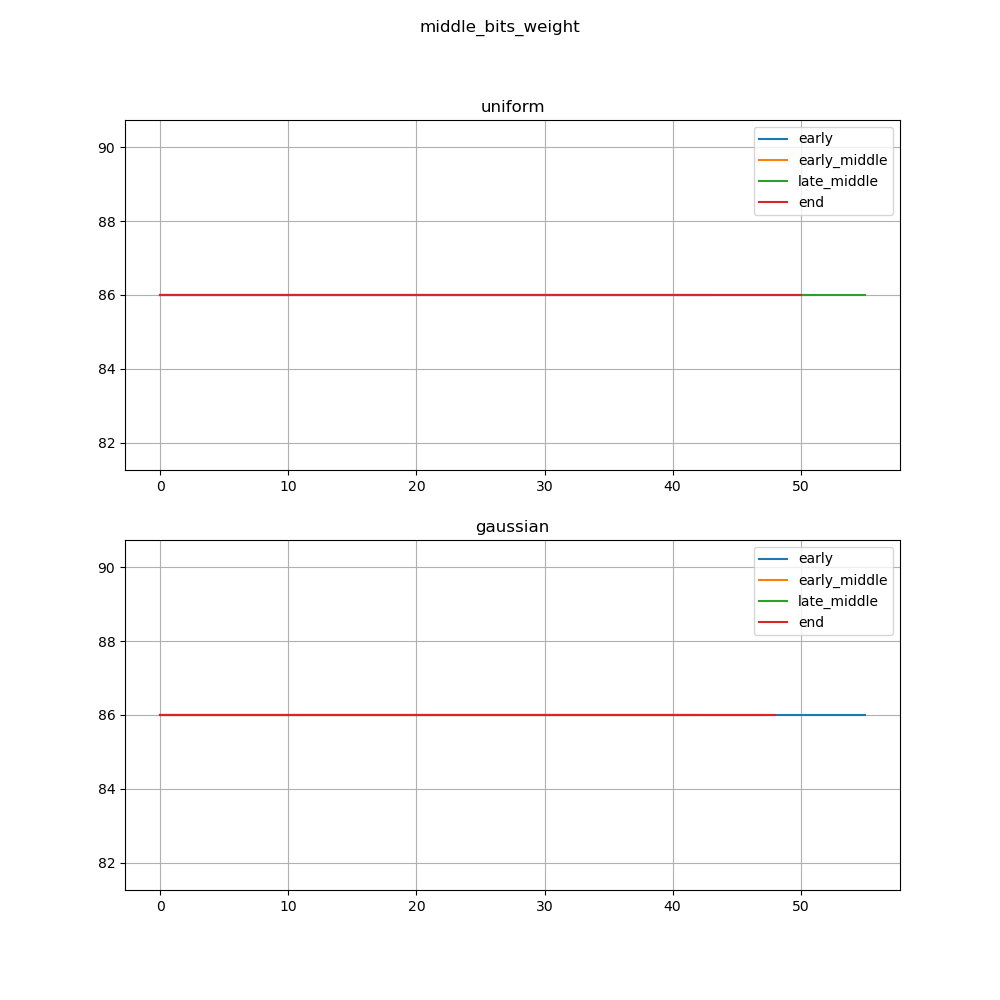
},

Memory, weights = {

Same but conv layers instead of BNs.

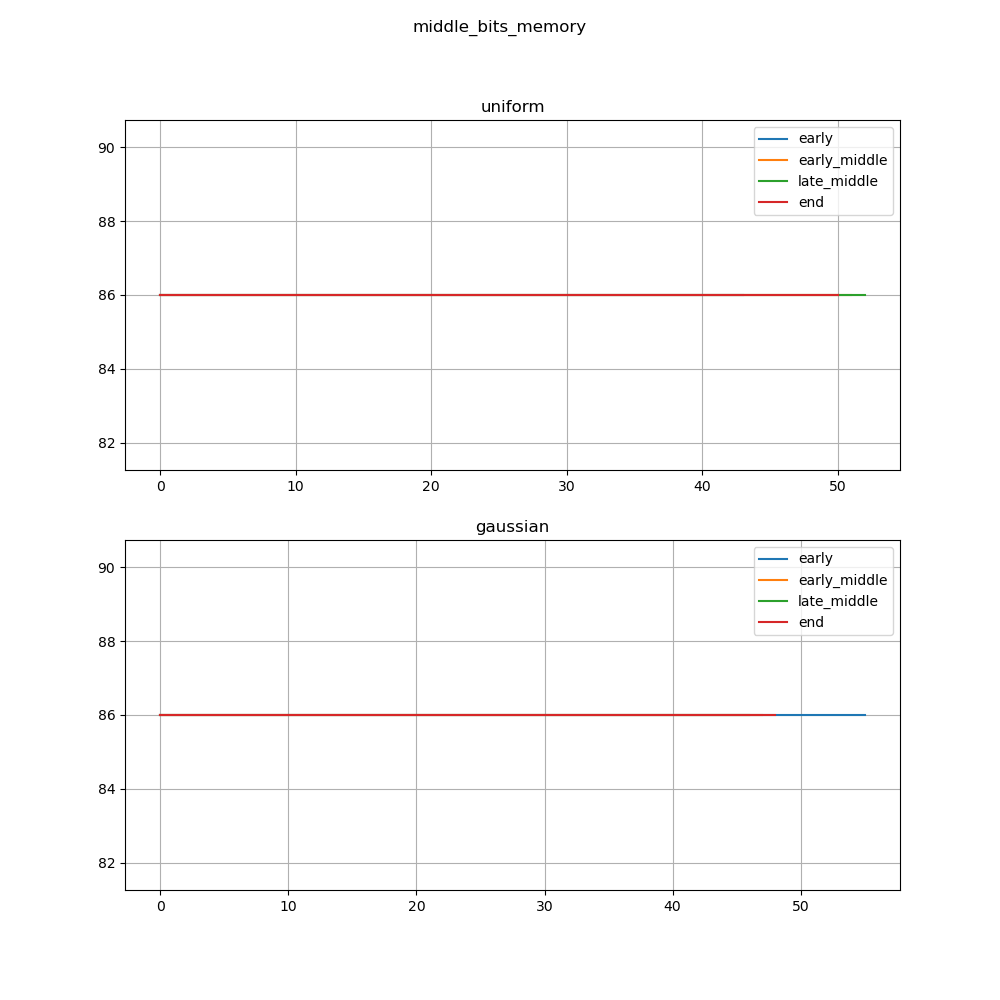
 }

}



We can see that we did not get a drop in accuracy in all targets.

This is due to the small change in the number (still a mantissa flip).



**Fault model to higher bit-flips:**

Fault\_target = bias/memory/weight

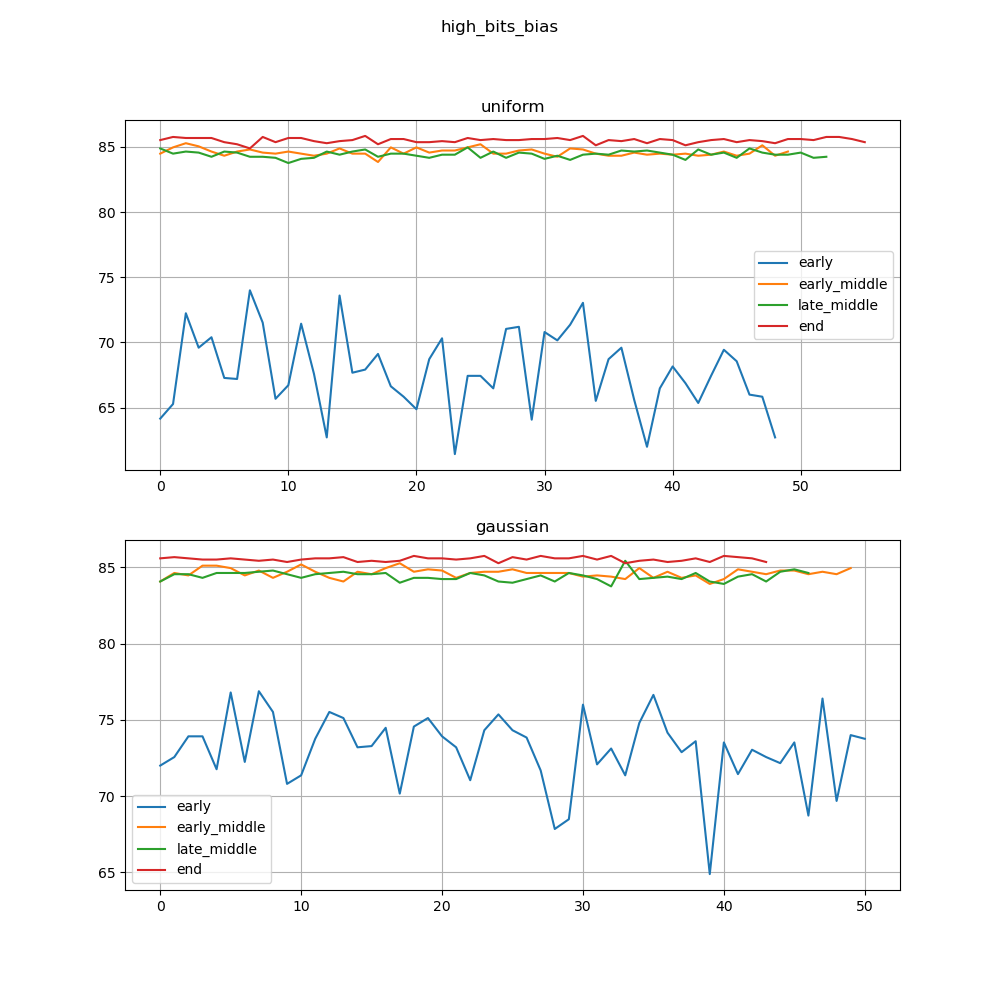
Number\_of\_faults = 20

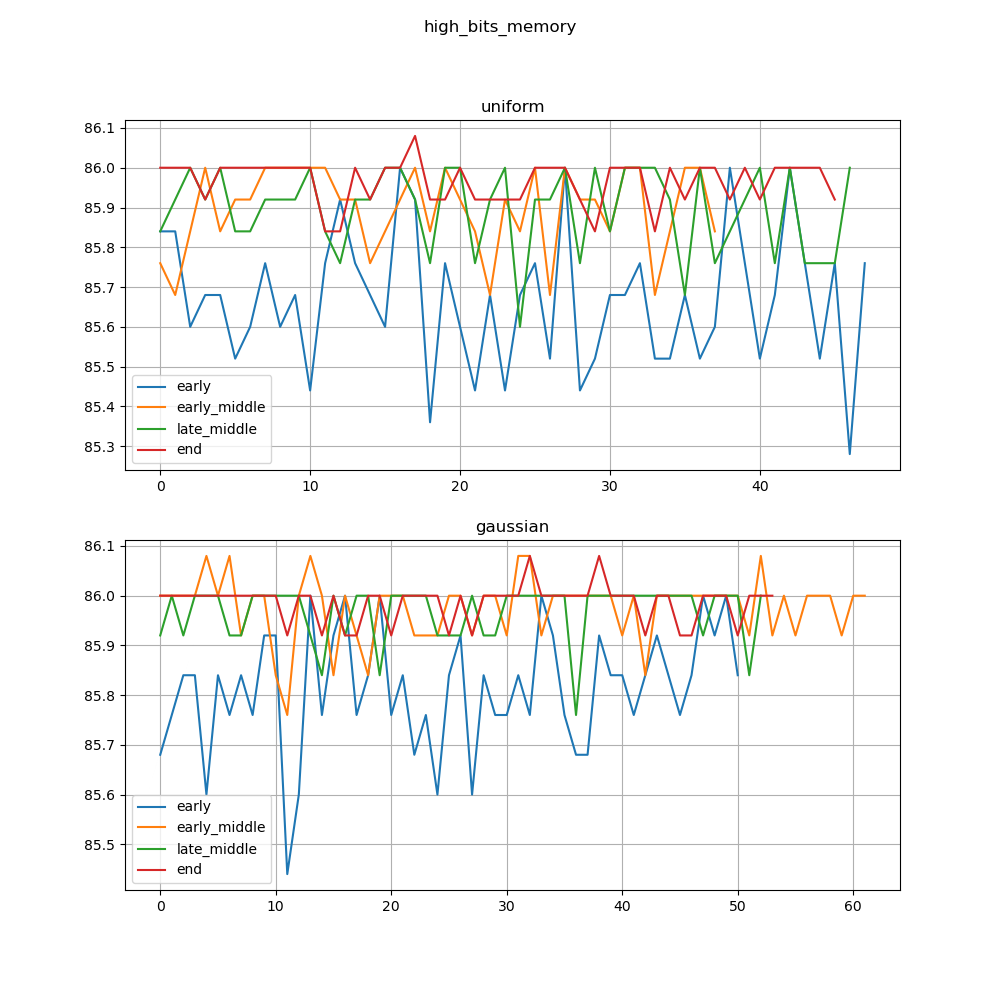
Num\_bits\_to\_flip = 5

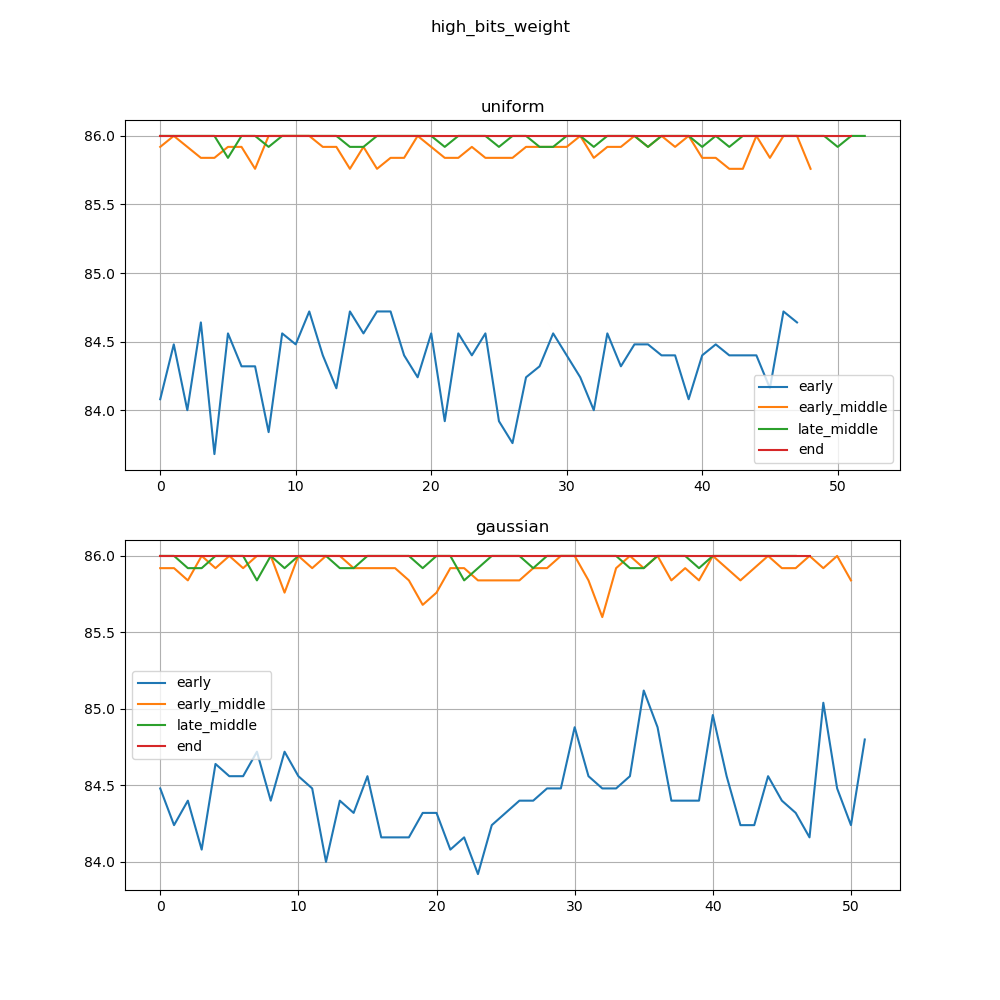
Bits\_to\_flip = 19, 20, 21, 22, 23

Fault\_distribution = gaussian/uniform

Target\_layer = same as before







We now begin to see drops in accuracy and thus FuSa violations.

This is because higher bits have higher impact on the value of the target.

We can see that bias faults have the most significant impact on FuSa violation numbers. This might be attributed to several reasons:

1. Bias is a more global value in context of a layer as opposed to weights. Weights are multiplied by part of the input (specifically in resnets, whereas bias is added to the whole feature map.
2. Role of the bias in a neuron: bias allows shifting of the activation function to the left or to the right, as opposed to weights where they determine the strength of the feature.
3. ReLU is very sensitive to bias value.

We can see that generally, attacking a layer at the start of the network is more prone to produce a FuSa violation.

DNN’s layers can be thought of as progressive, thus each group of layers focuses on a general task:

1. First layers: usually for low-level feature extraction, where each layer focuses on a simple pattern and propagates the result across the network. If faults were introduced into these layers, the network will struggle to extract those features, thus the error of the feature extraction will propagate to the end of the network, resulting in a big ripple effect,
2. Middle layers: usually to extract intermediate features by combining the features extracted from the first layers. in DNN’s, there are usually a lot of middle layers which work on similar features, thus there is a certain redundancy in these layers, so introducing faults into these layers will not have a big ripple effect.
3. Last layers: these layers are usually for high-level pattern extraction, thus as object detection, scenery detection etc.... Introducing faults into these layers will introduce FuSa violation, but now the question will be how much this fault affects the output, rather than a ripple effect like the first layers.

Please note that Gaussian distributed faults produce less FuSa violation in average as opposed to Uniform distribution.

This can be accounted to the small STD taken in the gaussian distribution, as the network is inherently built to be more robust to small perturbations due to learning from a very large dataset.

## Distribution of faults and its relation to FuSa violations

**Fault model:**

Fault\_target = bias/memory/weight.

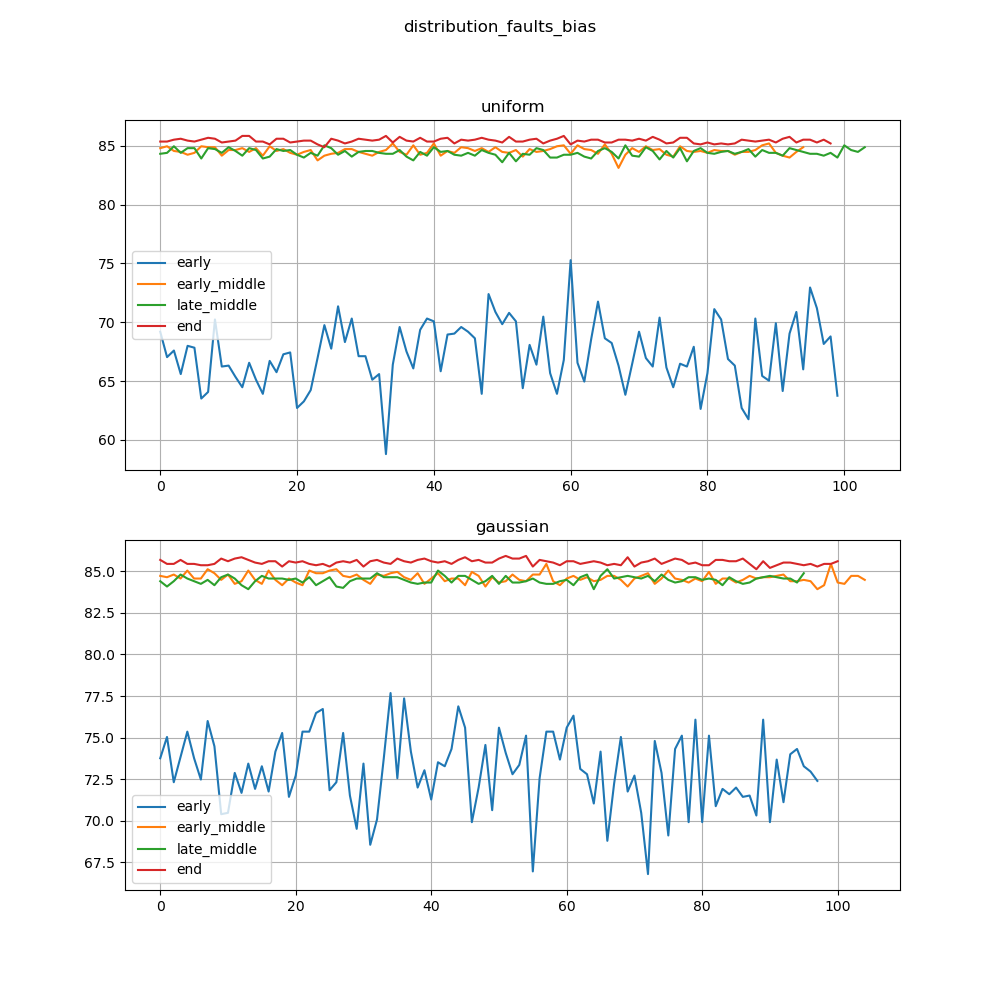
Number\_of\_faults = 20.

Num\_bits\_to\_flip = 5.

Bits\_to\_flip = 19, 20, 21, 22, 23.

Fault\_distribution = gaussian/uniform across target.

Target\_layer = same as before.



We can see that

Here we see a definite difference in the number of FuSa violations in Gaussian distribution of faults across fault target as opposed to Uniform distributed faults.

Weights in general determine the strength of influence of the input over the model output, thus by having a gaussian distribution for faults, we aggregate the faults in the mean region of the gaussian, thus if this part of the picture has very great influence, de facto the output will be influenced greatly.

Weights in a neural network fundamentally dictate the significance of the corresponding input features on the network's output. Hence, when faults are introduced into the weights following a Gaussian distribution, most of these faults will aggregate near the mean of the distribution, due to the nature of the Gaussian curve. If this region of the image where faults are aggregated corresponds to salient or highly influential features (since the influence is dictated by the weights), the overall network output could be significantly affected. The effect reached critical levels of 0% accuracy.

## Distribution of bit-flips and its relation to FuSa violations

**Fault model:**

Fault\_target = bias/memory/weight

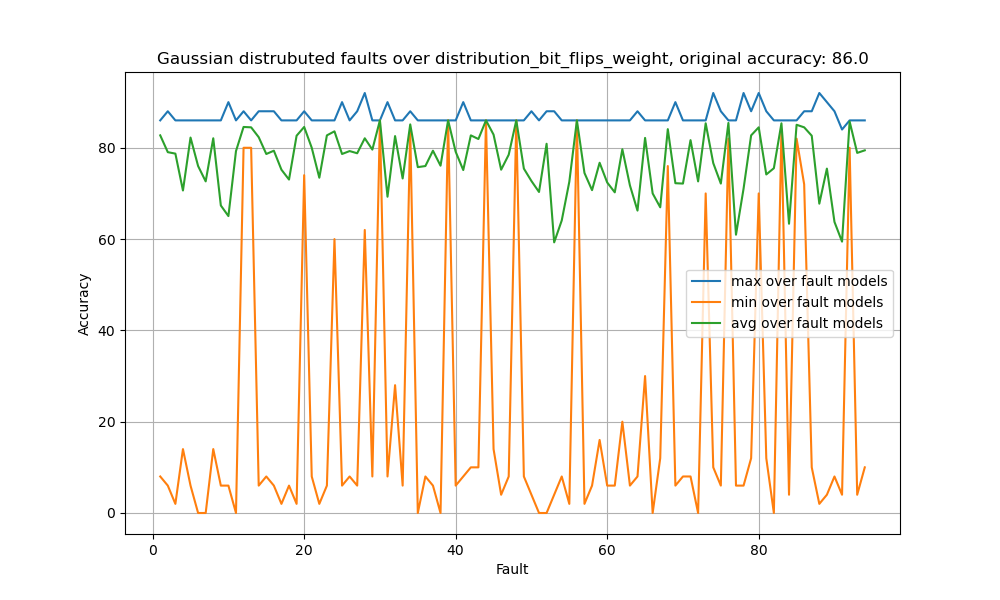
Number\_of\_faults = 20

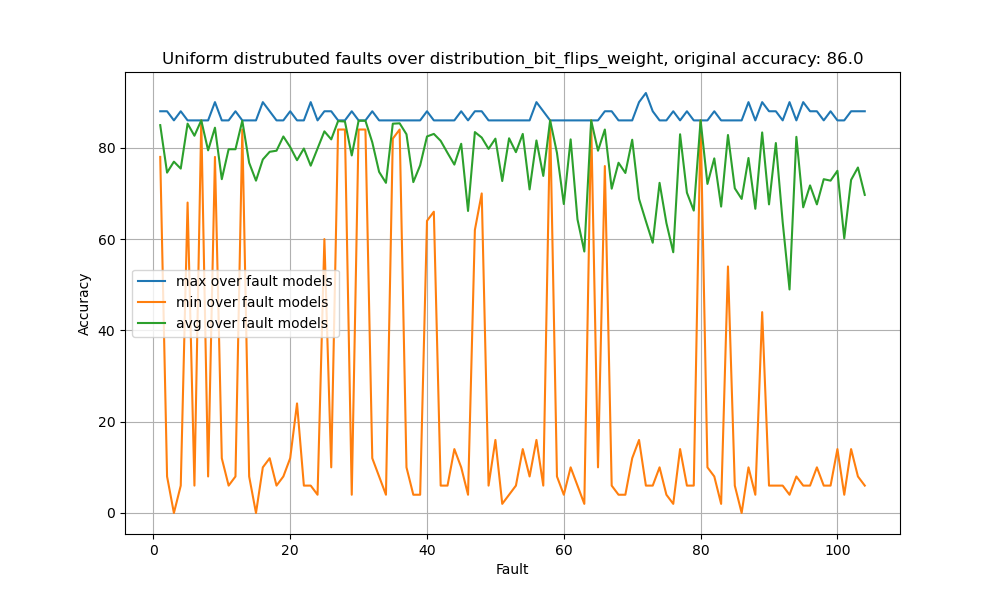
Num\_bits\_to\_flip = 5

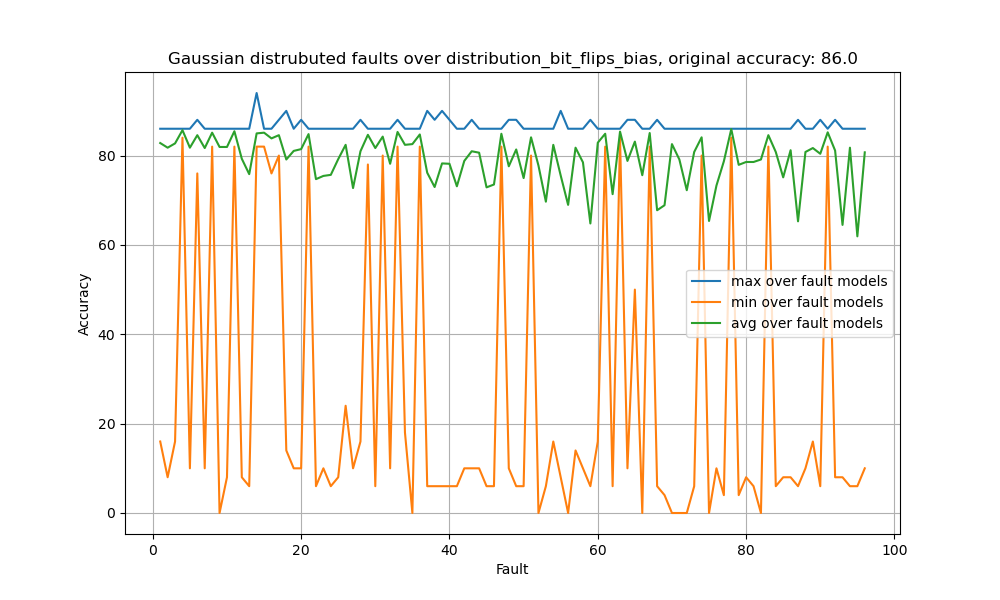
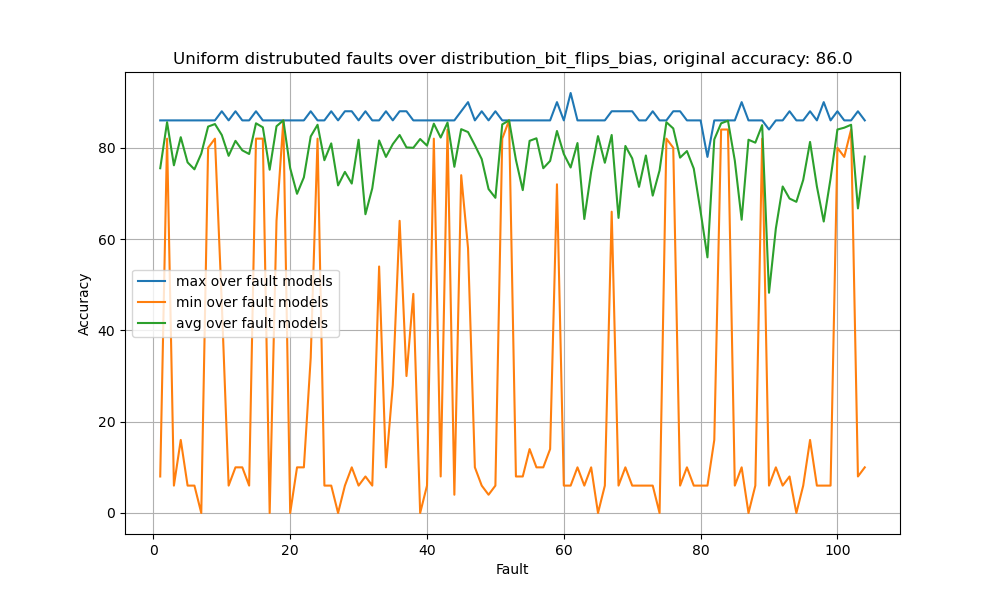
Bits\_to\_flip = randomized

Fault\_distribution = gaussian/uniform

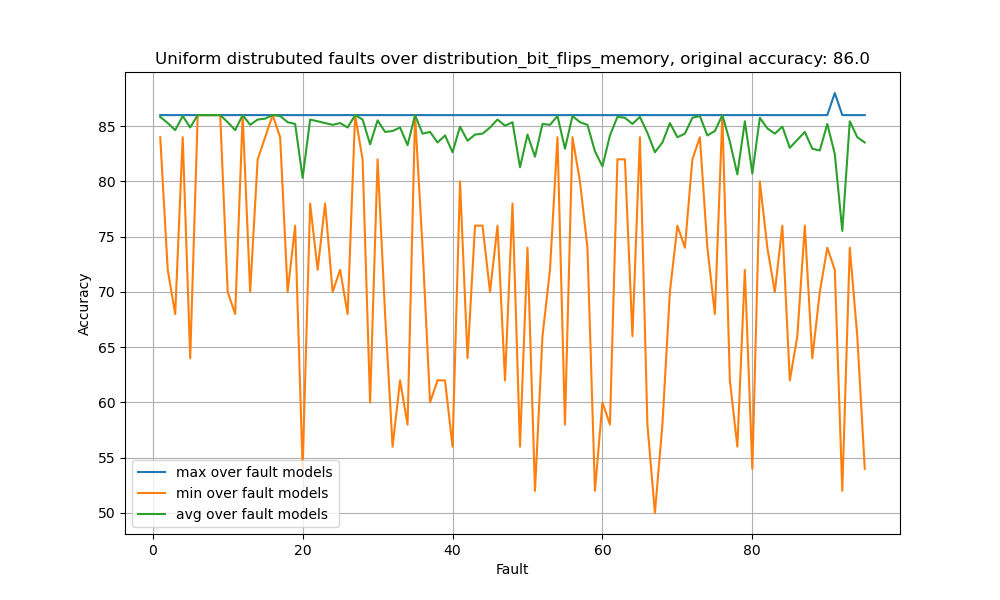
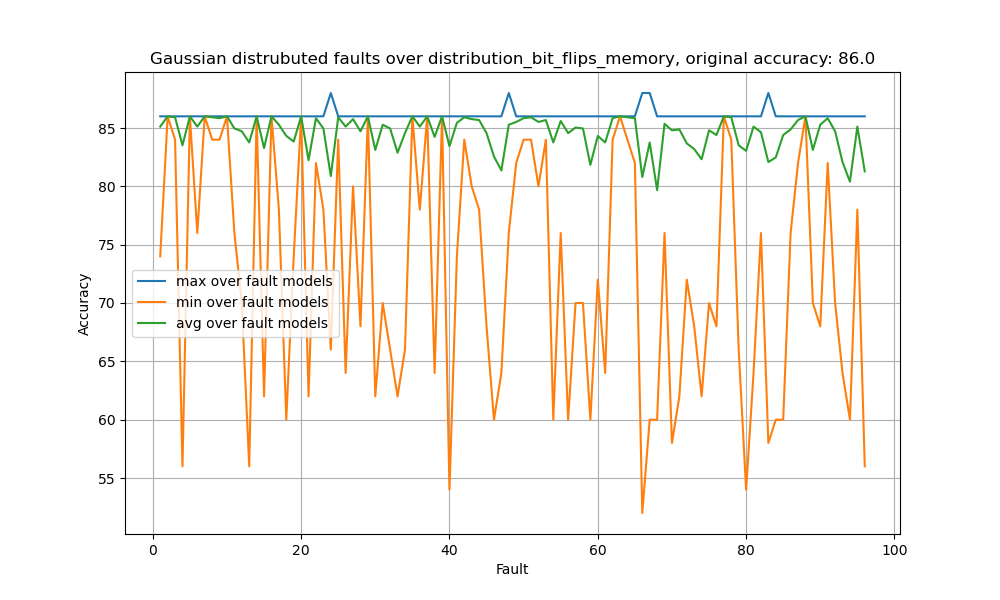
Target\_layer = randomized across each fault model

Note that the gaussian has a mean of 12 and std = 6



We can see that there is not much difference in behavior of these two distributions over bitflips on weights.

Same as on bias.



Same as weights and bias.

This can be attributed to the randomness of the choosing, where a fault could potentially change the decimal points as well as the integer of the float32, where this could be catastrophic for the model.

## Fault number and its relation to FuSa violations

**Fault model:**

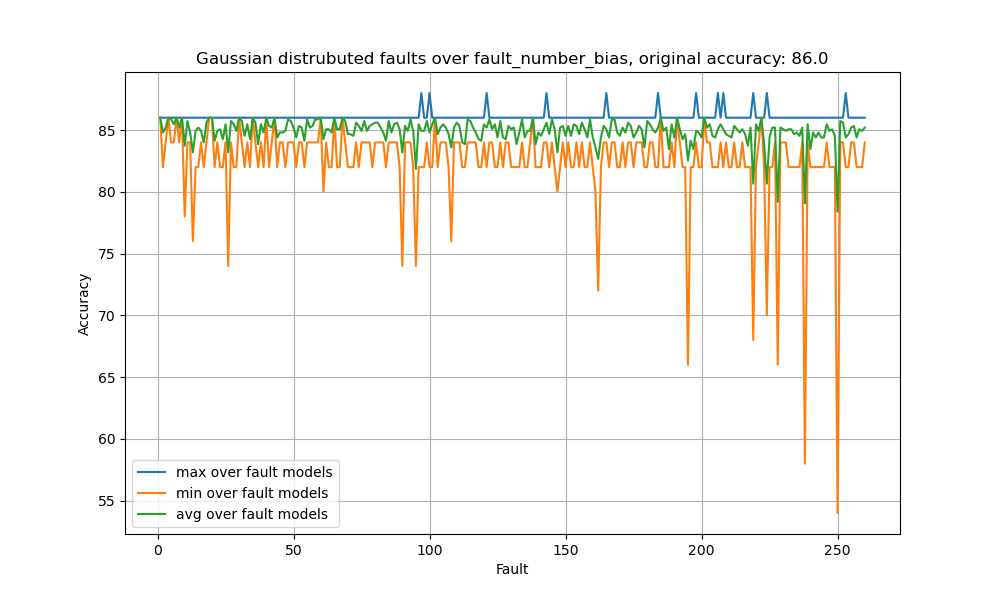
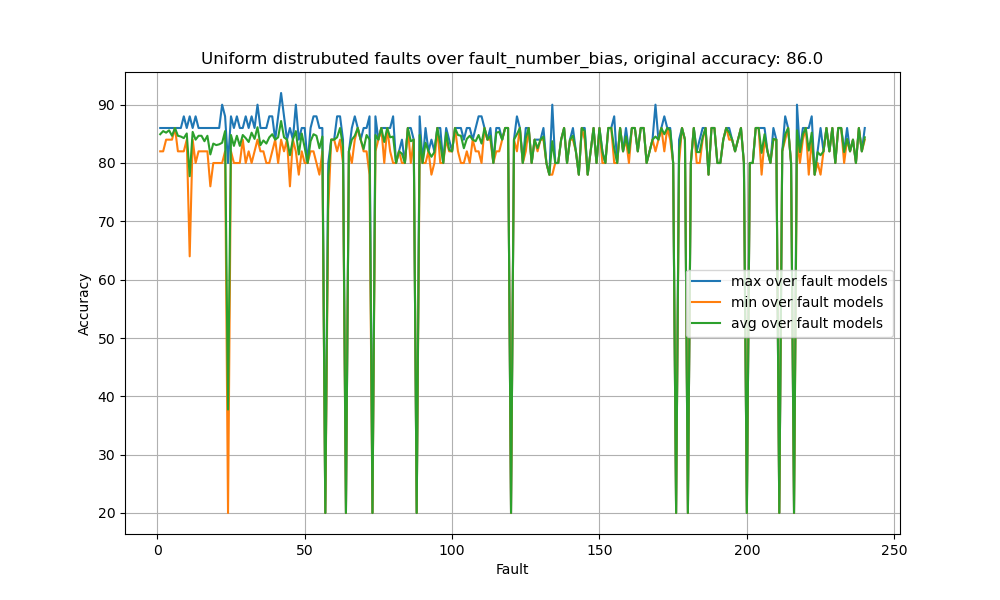
Fault\_target = bias/memory/weight

Number\_of\_faults = randomized from a given distribution (gaussian/uniform), min value is 0, max value is 500.

Num\_bits\_to\_flip = 5

Bits\_to\_flip = 19, 20, 21, 22, 23

Fault\_distribution = gaussian/uniform

Target\_layer = randomized across each fault model

Generally, we see that