A GRAYBOX DEFENSE THROUGH BOOTSTRAPPING DEEP NEURAL NETWORK

by

Kirsen Sullivan

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Statistics

West Lafayette, Indiana

December 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Bowei Xi, Chair
Department of Statistics

Dr. Mark Ward
Department of Statistics

Dr. Tom Sellke
Department of Statistics

Dr. Chris Clifton
Department of Computer Science

Approved by:
Dr. Jun Xie

To Andrew, Sloane and Heidi

ACKNOWLEDGMENTS

Purdue University's Department of Statistics and Graduate School helped fund this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Building a robust deep neural network (DNN) framework turns out to be a very difficult task as adaptive attacks are developed that break a robust DNN strategy. In this work we first study the bootstrap distribution of DNN weights and biases. We bootstrap three DNN models: a simple three layer convolutional neural network (CNN), VGG16 with 13 convolutional layers and 3 fully connected layers, and Inception v3 with 42 layers. Both VGG16 and Inception v3 are trained on CIFAR10 in order for bootstrapping networks to converge. We then compare the bootstrap NN parameter distributions with those from training DNN with different random initial seeds. We discover that the bootstrap DNN parameter distributions change as the DNN model size increases. And the bootstrap DNN parameter distributions are very close to those obtained from training with different random initial seeds. The bootstrap DNN parameter distributions are used to create a graybox defense strategy. We randomize a certain percentage of the weights of the first convolutional layers of a DNN model, and create a random ensemble of DNNs. Based on one trained DNN, we have infinitely many random DNN ensembles. The adaptive attacks lose the target. A random DNN ensemble is resilient to the adversarial attacks and maintains performance on clean data.

1. INTRODUCTION

1.1 Vulnerability in machine learning

In a resilient cyber physical system, the algorithms implemented in the software to process the data must be secured against adversaries as well as other elements of the cyber physical system. For example, an autonomous vehicle has cameras, radar and/or lidar sensors. The sensors receive information from the surrounding environment. The sensor data are processed to identify objects, classify these objects, and determine their distance and speed with regard to the vehicle. Each type of the sensors has its pros and cons. As machine learning (ML) and artificial intelligence (AI) gain widening implementation in transportation, some autonomous vehicle manufacturers prefer the less expensive camera sensors over the more expensive lidar sensors. However cameras are less reliable because the weather condition, the background of the objects and the image quality could potentially lead to a wrong decision and increase the odds of an accident. Therefore building resilient AI is crucial to process the image sensor data correctly and ensure the safety of self-driving cars.

Making AI resilient is an important task. Soon after AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012, which led to the third wave of AI, researchers discovered that deep neural network (DNN) can be fooled by adding targeted minor perturbations to clean images, i.e., the adversarial examples. Subsequently numerous attack algorithms were developed. This demonstrates the severity of AI vulnerabilities.

In this work, we focus on the full precision DNN models, because the large pre-trained models such as Inception models are very popular for transfer learning tasks, and a smaller convolutional neural network (CNN) is easy to train and capable of handling many classification tasks with high accuracy. Our approach does not involve a drastically different training process or model structure. We begin with a DNN model trained through the standard training process. Then a certain portion of the weights on a selected layer are replaced by random numbers generated from a Gaussian or Uniform distribution. The distribution parameters are determined through bootstrap. Bootstrapping several deep neural network models with different structure reveal that an increasingly large percentage of the bootstrap weight distributions and the bootstrap bias distributions become non-normal.

We utilize this information to form a graybox defense strategy against adversarial attacks. The adversary knows the trained model's structure and parameters. We build a random ensemble of the DNN models to classify the test data.

Since even a smaller CNN has a large number of parameters for each layer, replacing some weights by random numbers in a DNN model only slightly reduces the accuracy of the model on clean test data. Then an ensemble of DNN models with some random weights can achieve the same accuracy as the trained model without random weights. A random ensemble also achieves robust performance over adversarial examples. We further discover that randomizing the first convolutional layers in a DNN model produces the most robust results over adversarial examples, while randomizing the last several convolutional layers or the fully connected layers is less effective.

Given one trained DNN model, we can have infinitely many DNN random ensembles which all maintain the same accuracy over the clean test data as the trained model itself, and achieve robust performance over adversarial examples. We design this strategy to address the adaptive attacks. An adaptive attack will attack the defense strategy itself. By having infinitely many ensembles, we can frequently switch to a different ensemble without incurring additional training time. Thus the adaptive attack loses the target to attack. Although we cannot break the cycle of attacks and defenses, we can manage to stay ahead of the adversaries in the cycle.

2. LITERATURE REVIEW

2.1 Adversarial Learning

The most popular form of attacks against machine learning algorithms are evasion attacks. In this attack style, malicious adversaries generate adversarial samples with slight perturbations which can cause misclassification by a classifier. The perturbations can be small enough that they are not detectable to the human eye. The adversarial samples can also be used to evade detection by a learning algorithm at the test time. Examples of evasion attacks include spam emails, spoofing of biometric verification systems, and evading a deployed system at test time.

In the past several years, many evasion attack algorithms have been developed to generate adversarial samples that lead to misclassification by Deep Neural Networks (DNNs) in image classification. An evasion attack against DNN typically adds minor perturbation to the input to a DNN. In some cases, evasion attacks can be employed which creates minor perturbations to physical objects. This is much less common, and this review will focus only on digital attacks.

Evasion attack algorithms against DNN were first published in 2014 (Szegedy, Zaremba, Sutskever, *et al.*). Several survey articles are published providing the detailed timeline of adversarial attacks against DNNs, e.g., Yuan, He, Zhu, *et al.* [2]; Akhtar and Mian [3]; Biggio and Roli [4]: Akhtar, Mian, Kardan, *et al.* [5]. An influx of attack algorithms and effective defenses were devised in response to a 2017 NIPS competition organized by Google Brain. The winning teams' results are documented in Kurakin, Goodfellow, Bengio, *et al.* [6].

Attack algorithms are generated based on differing levels of the attacker's knowledge of the target system. Elements of the target system include the training data, the set of input features, and the full DNN structure including gradients, trained parameters, and hyper-parameters. In a white box attack, the attacker has full knowledge of all elements of the DNN under attack. A gray box attack may have knowledge of certain elements, such as the feature representation and the type of learning algorithm, but not of the training data or trained weights. In a black box attack, the attacker does not have knowledge about the DNN's structure, but may have access to confidence scores or labeled outputs. If the target

DNN is available as an oracle, black box attacks are often executed by querying the oracle. The attacker then uses the confidence scores or labels returned by the oracle to train a local model and generates adversarial samples against the local model.

Beyond the attacker's knowledge, the attacker's goal and ability to manipulate input data further define the optimization problem which the attack strategy aims to solve. There are several security violations which may be the aim of the attacker. These include privacy violations, integrity violations and availability violations. Privacy violations aim to obtain private information about the data, system or users. An integrity violation is one in which detection is evaded and normal operation of the system is maintained. An availability violation aims to make the system malfunction for the legitimate users. A second class of attack goal is error specificity. The aim here is to cause the system to misclassify input. The goal can be either targeted or untargeted, where in the former case, a specific (incorrect) target classification is desired. An untargeted attack simply wishes to make the classifier select any incorrect class. In a final category of attack goals, the aim is to maximize attack specificity. As with error specificity, this specificity can be either in terms of a targeted set of inputs (e.g., a specific user), or indiscriminately applied to all input samples. The capability of the attackers also help define the optimization function. An attack which can influence both training and test data is sometimes known as a poisoning, backdoor or trojaning attack [7]. In an evasion attack, the attacker can only influence the test set.

2.1.1 White Box Attacks

L-BFGS

Evasion attack algorithms were first reported in [1]. Adding perturbation δ to an image x can cause a DNN to misclassify the adversarial image, even when the perturbation is imperceptible. This non-random perturbation is found by maximizing the DNN's prediction error. For a given image x of dimensionality m and a target class label t, δ was obtained by finding an approximate solution to the following box constraint optimization problem using L-BFGS.

$$\min \|\delta\|_2 \qquad \text{s.t. } C(x+\delta) = t,\, x+\delta\,\epsilon\,[0,1]^m \tag{2.1}$$

The paper also observed the transferability of adversarial images to other networks. The specific adversarial image can be misclassified by a second DNN which was trained using a different subset of the data.

Fast Gradient Sign Method (FGSM) and its variations

As a one-step attack, FGSM has a low computation cost and can find an adversarial image quicker than L-BFGS. Let $J(x, y)$ be the cross-entropy cost function and y be the true label of an image x. Using the sign of the gradient of the cost function, Goodfellow, Shlens, and Szegedy [8], generated adversarial image $x + \delta$ with perturbation δ as

$$\delta = \epsilon * sign(\nabla_x J(x, y)) \tag{2.2}$$

Since the initial introduction of FGSM, several modifications have been presented. By using the scaled gradient instead of the sign of the gradient, Rozsa, Rudd, and Boult [9] produced more diverse adversarial images. (Tramèr et al., 2018) suggested a small randomized single-step attack. Instead of a simpler one step attack, adversarial samples can be generated by iteratively following the direction of the gradient while clipping the generated image to stay inside an ϵ-neighborhood of the original image x as in the iterative FGSM (I-FGSM) in Kurakin, Goodfellow, and Bengio [10]. Projected Gradient Descent (PGD) attack is an I-FGSM with many random starting points in the L_∞ ϵ-neighborhood. This yielded stronger adversarial samples, and is the strongest adversarial attack which utilizes only first order information about the DNN (Madry, Makelov, Schmidt, *et al.* [11]). Further attacks have been developed which add momentum to an iterative FGSM (MI-FGSM) to boost attack strength and maintain transferability of adversarial images. The update with momentum is the following,

$$h_{t+1} = \beta * h_t + \frac{\nabla_x J(x_t, y)}{\|\nabla_x J(x_t, y)\|_1} \tag{2.3}$$

$$x_{t+1} = x_t + \alpha * sign(h_{t+1}) \tag{2.4}$$

Where $\alpha = \epsilon/T$, T being the predetermined number of iterations (Dong, Liao, Pang, *et al.* [12]). This method has been applied to an ensemble of DNNs by averaging the logits of DNNs in the ensemble to compute the cross-entropy $J(x, y)$.

Carlini and Wagner (C&W)

C&W attack (Carlini and Wagner [13]) is a modification of L-BFGS attack. In order to address the issue of the classifier constraint in L-BFGS attack being highly non-linear, C&W modify the objective function making it better suited to optimization. Their L_2 attack is sufficiently strong to bypass a number of detection and defense methods. They solved the following box constraint optimization problem to find an adversarial perturbation δ.

$$minimize \left\| \frac{1}{2}(\tanh(w) + 1) - x \right\|_2^2 + c * f(\frac{1}{2}(\tanh(w) + 1) \tag{2.5}$$

With f defined as

$$f(x', t) = \max\left(\max\{Z(x')_{\mathrm{i}} : \mathrm{i} \neq t\} - Z(x')_{t'} - \kappa\right) \tag{2.6}$$

where $Z(x)_{\mathrm{i}}$ is the output of the softmax for class i. After optimization, δ is retrieved from w by the following change of variable.

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x \tag{2.7}$$

Inspired by the C&W attack, Elastic-net attacks (EAD) in Chen, Sharma, Zhang, *et al.* [14] used the same loss function $f(x, t)$, but introduced elastic-net regularization. This further encouraged similarity to the original image. EAD's L_1 attack achieved comparable performance as the C&W L_2 attack.

Jacobian-based Saliency Map Attack (JSMA)

JSMA computed the Jacobian of either the logits as in Papernot, McDaniel, Jha, *et al.* [15] or the outputs of the softmax as in Papernot, McDaniel, Jha, *et al.* [15] for an image x, and built a saliency map. This saliency map was used to characterize the input-output relation of DNNs. From there, the algorithm is a greedy attack algorithm which iteratively updates the most influential pixel to generate adversarial images. This method is an effective attack which only modifies small portions of the input image. However, it has a high computational cost.

DeepFool

Moosavi-Dezfooli, Fawzi, and Frossard [16] introduced an iterative attack algorithm to efficiently search for adversarial samples. At each iteration, a classifier $f(x_t)$ was linearized at the current point x_t and an update was computed by finding the closest projection of x_t on a hyperplane of the adversarial classes. DeepFool attack can also be used to find a universal adversarial perturbation that causes almost all images to fool a classifier (Moosavi-Dezfooli, Fawzi, Fawzi, *et al.* [17]).

CPPN EA Fool

Compositional pattern-producing network encoded evolutionary algorithm (CPPN EA) [18] can be classified as a false positive attack. The adversarial images are designed to be unrecognizable to humans as having been tampered with. Adversarial images generated by this method were accepted into an art competition at a rate of 35.5% [2]. The authors were able to show that this method can identify significant features similar to JSMA, but with less computational requirements.

Hot/Cold

While FGSM considers the gradient of the loss function, the Hold/Cold algorithm of Rozsa, Rudd, and Boult [9] considers earlier layers, particularly the layer immediately preceding the softmax layer. In CNNs this layer is a fully connected linear layer with layer outputs still corresponding to class assignments. An inverted one-hot vector is applied at this penultimate layer, with the goal of creating both features in the earlier layers which correspond to the designated adversarial (hot) class and features which decrease the likelihood of being associated with the non-hot classes. In addition to the 'hot' class, a 'cold' class is introduced in the penultimate layer. Now the imposed feature vector for the penultimate layer is defined for the hot class: the absolute value of the network's feature output for the hot class; for the cold class: the negative value of the network's feature output for the cold class; and 0 for all other classes. Using backpropagation, a new image is computed with the necessary changes needed to move the input image from the original (cold) class to the new desired (hot) class.

The optimization to compute the adversarial example is then performed using the so-called PASS method. The Perceptual Adversarial Similarity Score (PASS) measure created by [9] acts as a similarity measure which quantifies how adversarial a misclassified image is.

Natural GAN

Zhao, Dua, and Singh [19] generated adversarial images and texts using Generative Adversarial Networks (GANs). This has the advantage of making the adversarial examples, more natural, for example, the texts produced follow semantic rules. Other methods search for some adversary x^* which is similar to x but produces a different classification. The goal of natural GAN is to search for an x^* which meets the above criteria, but also fits the underlying data distribution P_x. This is done by finding an adversary in the P_x space, and then mapping it back to x^* with the help of a generative model, G. G is trained to map random noise back to the input distribution.

$$\min_{z} \parallel z - P_x \parallel \tag{2.8}$$

$$s.t.\ f(G(z)) \neq f(x) \tag{2.9}$$

2.1.2 Black Box attacks

Model-based Ensembling Attack

Liu et al proposed a Model-based Ensembling Attack for targeted adversarial images [20]. The authors argued that it is more difficult to targeted adversarial images are less transferable over deep models than non-targeted adversarial images. To that end, they designed a transferable targeted attack technique which can be used in black box attacks, or attacks in which the CNN structure is unknown. The targeted adversarial example is derived using k deep networks by the following optimization problem.

$$\arg\min_{x'} \ -\log((\sum_{\mathrm{i}=1}^{k} \alpha_{\mathrm{i}} f_{\mathrm{i}}(x', l'))) + \gamma \|x' - x\| \tag{2.10}$$

where f_{i} is the function of each of the k networks and α_{i} is the weight of the i^{th} ensemble (weights sum to 1). x is the input image and l' is the target class.

Zeroth Order Optimization (ZOO)

ZOO Chen, Zhang, Sharma, *et al.* [21] is a modification of C&W's L_2 norm attack, under the assumption that both the gradients and model structure are unknown to the attacker. The loss function in ZOO replaces the output of the softmax in C&W's loss function with the log output of the DNN. The optimization on the loss function does not require knowledge of the gradients, instead it estimates them using the symmetric difference quotient, implementing stochastic gradient descent. This adds greatly to the computation cost compared with C&W.

One-pixel

The One-pixel attack proposed in Su, Vargas, and Sakurai [22] operated under the constraint of changing the value of a single pixel of an image to fool a DNN. These perturbations are based on differential evolution. Differential evolution does not require a differentiable optimization problem, thus it can be employed in a black box scenario where the attacker only knows confidence scores.

Surrogate Model & Real World Attacks

White box attacks can be employed in black box environments with some success due to the transferability of adversarial images to models on which they were not trained. The scenario in which the attacker does not have full knowledge of the DNN is more likely in real world attacks. Kurakin, Goodfellow, and Bengio [23] showed that adversarial images can be generated using the predicted labels in FGSM instead of the true labels. (Papernot et al., 2017) trained and generated adversarial images against a local substitute model using probes. With 800 queries sent to Amazon Web Services and Google Cloud Prediction, most of the adversarial samples were misclassified by the target models hosted by Amazon and Google. Liu, Chen, Liu, *et al.* [20] generated a targeted attack against k networks in the white box fashion, and showed the adversarial images can transfer to an additional black box network. Ilyas, Engstrom, Athalye, *et al.* [24] further developed a black box attack under limited queries and partial output knowing only the top k class probabilities. Unlike previous attacks against image level classifiers, Liu, Yang, Liu, *et al.* [25] developed a black box attack against state-of-the-art object detectors.

Adversarial poisoning and Backdoor attacks

There are multiple ways to train a hidden, unexpected classification behavior into a CNN. First, a bad actor with access to the CNN can insert an incorrect label association (e.g. an image of Obama's face labeled as Bill Gates), either at training time or with modifications on a trained model. We consider this type of attack adversarial poisoning. In contrast, we define a CNN backdoor as a hidden pattern trained into a CNN, which produces unexpected behavior if and only if a specific trigger is added to an input. Such a backdoor does not affect the model's normal behavior on clean inputs without the trigger.

In the context of classification tasks, a backdoor misclassifies arbitrary inputs into the same specific target label, when the associated trigger is applied to inputs. Inputs samples that should be classified into any other label could be "overridden" by the presence of the trigger. In the vision domain, a trigger is often a specific pattern on the image (e.g., a sticker), that could misclassify images of other labels (e.g., wolf, bird, dolphin) into the target label (e.g., dog). It is important to note that adding the same backdoor trigger causes arbitrary samples from different labels to be misclassified into the same target label.

Gu et al. proposed BadNets, which injects a backdoor by poisoning the training dataset [26]. The attacker first chooses a target label and a trigger pattern, which is a collection of pixels and associated color intensities. Patterns may resemble arbitrary shapes, e.g., a square. Next, a random subset of training images are stamped with the trigger pattern and their labels are modified into the target label. Then the backdoor is injected by training CNN with the modified training data. Since the attacker has full access to the training procedure, she can change the training configurations, e.g., learning rate, ratio of modified images, to get the backdoored CNN to perform well on both clean and adversarial inputs. Using BadNets, authors show over 99% attack success (percentage of adversarial inputs that are misclassified) without impacting model performance in MNIST [26].

Another approach (Trojan Attack) was proposed by Liu et al. [27]. They do not rely on access to the training set. Instead, they improve on trigger generation by not using arbitrary triggers, but by designing triggers based on values that would induce maximum response of specific internal neurons in the CNN. This builds a stronger connection between triggers and internal neurons, and is able to inject effective (¿ 98%) backdoors with fewer training samples.

Backdoor attacks and adversarial poisoning assume the attacker has modification level access to the training set and/or the trained CNN. In contrast, our work focuses on evasion attacks, i.e., assumes a clean training set. The attacker has full knowledge of training set, model structure and parameters, etc., but does not have editing access of these features.

2.1.3 Existing Defense Strategies

Naturally, many defense strategies were developed to counter the attack algorithms. It is unfortunate that developing resilient AI proves to be extraordinarily difficult. A defense strategy could reduce the effectiveness of existing attacks. But a newer adaptive attack fine tuned to target the key components of the defense strategy could discover the vulnerabilities of the defense and render the defense less effective. This seems to become a never ending cycle of attacks and defenses.

Although currently there is no perfect defense which is able to correctly classify the adversarial examples generated by the existing attacks and the adaptive attacks, several defense strategies are shown to provide a degree of robustness. We summarize them as follows.

Adversarial training

There are several variants of adversarial training. In general the procedure includes adversarial examples in the training process. As the network is being trained, additional adversarial images are created at each step. This technique was found to lead to the learner overfitting the generated adversarial examples, a phenomenon known as 'label leaking'. One approach to avoid this is to introduce an ensemble strategy. The adversarial examples are generated against other pre-trained models rather than the learner being trained [10].

Network distillation

In another adversarial training technique, called defensive distillation (Papernot, McDaniel, Wu, *et al.* [28]), one model is trained to predict the output probabilities of another model which was trained on an earlier, baseline standard. The first model is trained with hard labels, the second is trained with the output from the first model, or 'soft' labels.

Adversarial training is a flexible strategy which can be combined with another defense strategy. It has been shown that adaptive attacks can significantly reduce the effectiveness of this training method. (Tramer, Carlini, Brendel, *et al.* [29])

Adversarial detecting

Many researchers have focused on attempting to detect adversarial examples at the testing stage [30]–[34], [35]–[39]. This is done through a variety of methods. Some trained deep neural networks with two classes: adversarial or clean. Others used the original classes, but also added an additional outlier class. Others compare the distribution of the images to the distribution of clean images, or the uncertainty of input images to that of clean data.

Image Pre-processing

In 2017, Google Brain organized a Competition on Adversarial Attacks and Defenses [6]. Several winning teams applied pre-processing techniques, such as JPEG compression and image denoising, to adversarial examples to reduce the impact of adversarial perturbations, which are high frequency noises to the human eyes. Subsequently other pre-processing techniques were proposed too. It is also a flexible strategy which can be combined with another defense strategy. However if the adversary is aware of the pre-processing technique, an adaptive attack is able to reduce its effectiveness.

Different model structures

One example of a different model structure is binarized neural network (BNN), where activations and weights only use binary values. When a neural network model is implemented in a cyber physical system, accuracy may not be the only concern. For example, autonomous vehicle requires a resilient, fast, and energy efficient algorithm to process the large amount of sensor data. There are three different types of hardware computation platforms. While GPU or TPU is typically used for CNN, BNN is a suitable candidate for field-programmable gate array (FPGA). Binarized networks have been shown to be competitive with full-precision models [40] and has demonstrated to use less battery power and less time. Meanwhile, BNN is relatively robust [41]. Adversarially trained BNN can achieve reasonable accuracy over adversarial examples.

3. BOOTSTRAP CNN

The random ensemble defense strategy described in Chapter 4 begins with a DNN model trained through the standard training process. Then a certain portion of the weights on a selected layer are replaced by random numbers generated from a Gaussian or Uniform distribution. The distribution parameters are determined through bootstrap. Bootstrapping several deep neural network models with different structure reveal that an increasingly large percentage of the bootstrap weight distributions and the bootstrap bias distributions become non-normal. This chapter describes the network architectures, bootstrapping strategies and contains results pertaining to the NN weight distribution properties.

3.1 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN) are a class of neural networks which has been used heavily in image analysis. CNN architecture includes a number of convolutional layers, as well as pooling and fully connected layers. In a convolutional layer, a grid is convolved across the width and height of an image. This grid, or filter, performs a dot product between the filter and the input. As the network learns, this kernel can extract features from images. The output feature maps can then be passed on to more convolutional layers for higher levels of feature extraction. Pooling layers are used to down sample the output feature map size as well as increase local translational invariance. Finally, the output feature map is flattened and a fully connected layer is implemented to classify the images.

During training, the network is learning the kernel parameter values. There are also predetermined hyperparameters for a convolutional layer, including kernel size, padding, and stride. Kernel size refers to the grid size that is being convolved over the input. Typical grid sizes are 3x3 and 5x5, but they may also be one-dimensional or three-dimensional. Smaller kernel sizes are able to detect features within a smaller local area, whereas larger kernel sizes may be useful for extraction of larger features. Smaller kernel sizes also decrease the total number of trainable parameters in the network, decreasing computational cost associated with training the model. As a 3x3 kernel slides over the image, the outermost pixels will never be in the center of the kernel. A way to address this is to introduce padding to the outer edge of the image. Stride is another hyperparameter, which determines how far apart kernels are from one another in an image. A typical stride value is one, but sometimes larger values are used in order to downsample the output feature map size.

A convolution is ultimately a linear transformation. So even if many convolutions are stacked on top of each other, it would simply reduce to a linear transformation without some non-linearity being introduced into the network. A nonlinear activation function after the convolutional layer achieves this and greatly increases the complexity of tasks the network can learn. One important activation function is rectified linear unit, or ReLU. This function reassigns all negative values to 0.

The output feature maps from the convolution and activation function are sensitive to the local position of the feature. Pooling is implemented to down sample the feature map and increase the local translational invariance. Max pooling and average pooling are both common techniques and are used in the networks discussed below. Pooling is used to down sample feature maps by summarizing information in a local region. Max pooling selects the highest pixel value within a specific kernel size and applies it to all pixels in the kernel. The effect creates sharp contrast between features and emphasizes light features on dark backgrounds. Average pooling is another useful pooling technique. Pixel values are smoothed as values take on the average value of their close neighbors. This pooling method may not be as successful at identifying sharp features as max pooling. The kernel size is most commonly 2x2, with a stride of 2. Thus, the output of the pooling layer is one-quarter the size of its input feature map.

Convolutional layers are designed to decrease the chance of overfitting by greatly reducing the number of parameters as compared with fully connected architectures. Pooling further reduces the number of potential parameters. An additional layer known as the dropout layer is often employed before the final output layer to further reduce the risk of overfitting. During the dropout layer, each node has a certain predetermined probability, p, of being set to 0 or 'dropped'. Typical dropout rates are around 0.5. The nodes which are not set to 0 are scaled up to 1/(1-p), so the sum over all nodes is unchanged.

The final step of a CNN is the classification function, performed by a fully connected layer. The output of the convolutional layers is flattened and then passed to a linear transformation.

For each image, a value is output for each of the classes in the dataset. A softmax layer can be applied following the linear transformaion. The softmax function normalizes the output to a probability distribution over the potential classes. These probabilities are proportional to the exponentials of the output.

CNNs are trained by sending the inputs through the network, known as the forward pass, followed by a backward pass, or back propagation. In the backward pass, the CNN adjusts its parameters proportionate to the error in its guess. It does this by traversing backwards from the output, collecting the derivatives of the error with respect to the parameters of the functions (gradients), and optimizing the parameters using gradient descent. There are many important optimization techniques utilized, but only two are employed in the architectures considered in this work. The first is Stochastic Gradient Descent (SGD), and the second is Adaptive Moment Estimation, or Adam Kingma and Ba [42]. SGD updates the parameters by following the gradient, with updates happening in mini-batches. There is one pre-determined, static learning rate which is used throughout training. Due to the algorithm being sensitive to the order of the data, for each epoch of training, the training data should will be randomly shuffled.

The Adam algorithm estimates an adaptive learning rate for all parameters involved in the training of gradients. It is computationally more efficient than SGD. Rather than a fixed learning rate, the rates are different for each parameter and are based on the average first moment and the average of the second moments of the gradients. Adam is a popular technique because it is fast and achieves good results.

As with other optimization problems, CNNs are sensitive to initialization, and may not be able to converge with poor initialization. For many well-known neural networks, pre-trained weights are available which have been trained on the ImageNet dataset. Using these pretrained weights greatly increases training speed, even when using a novel dataset. Without pretrained weights, another initialization technique is required. In 2015, He, et al. [43] introduced a random weight initialization scheme which takes into account the non-linearity of the ReLU function. Weights are initialized according to a zero-mean Gaussian distribution with standard deviation $\sqrt{(2/n)}$, with n being the number of connections in the layer.

An adaptation using a Uniform distribution has also been developed, known as the Kaiming uniform method.

A number of CNN attack and defense strategies were introduced in Chapter 2. Adaptive attacks make it particularly difficult to implement robust defense strategies [29]. Our approach is to build a random ensemble to address adaptive attacks. The ensemble of randomized learners, discussed in detail in Chapter 4, requires a certain portion of the weights from a selected layer to be replaced with random numbers from either a Uniform or Gaussian distribution. The distribution parameters of the weights in a CNN are unknown, but can be estimated by bootstrapping. All networks were trained using the PyTorch framework in Python, and with GPU computing resources.

3.2 Inference for CNN Parameters

The effort to understand the theoretical properties of CNN models can trace back to the late 1980s. Interesting there are limited work on bootstrap CNN [44]–[54] and also very limited work on understanding CNN parameter distributions [55]–[61].

Boostrapped NNs were often used to build an ensemble to enhance the performance. Bootstrapping a NN was also conducted to quantify the uncertainty of NN output. In the literature, bootstrapping a NN was never intended to study NN parameter distribution.

Several papers said that NN weights have heavy tails during the training process. These work did not take into consideration of the randomness of the training data-set. We discovered that for NN models with smaller model structure, the bootstrap NN parameter distributions are mostly normal. For bigger and more complex NN models, a percentage of the bootstrap NN parameter distributions are still normal, although as the model size and complexity increases, more bootstrap parameter distributions becomes heavier tailed than normal.

3.3 Data and Bootstrapped CNN Models

3.3.1 Benchmark Datasets

Two datasets are utilized in the bootstrap experiments. Both datasets are available as built-in PyTorch subclasses of the class torch.utils.data.Dataset. The first is MNIST, a database of handwritten digits. There are a total of 70000 black and white images of size 28 x 28. The database includes 60,000 training images and 10,000 test images. There are ten classes, one for each of the digits 0 to 9, inclusive. The digits in the training set were written by approximately 250 writers. Half of the digits were written by Census Bureau employees, the other half by high school students. The digits have been size-normalized and centered in the image. The data is in four GZ files, one each for training images, training labels, test images and test labels.

The second dataset used for the CNN bootstrap experiments is CIFAR-10 [62]. This dataset consists of 60,000 32x32 color images. There are a total of 10 classes, with 6000 images per class. The images are divided as 50000 training images and 10000 testing images. The ten classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The data is available as a GZ archive file. The archive contains five files with train image data and one file of test image data. Each of these files are Python 'pickled' objects. These files contain a dictionary with two elements. The first, 'data', is a numpy array of size 10000x3072. This contains pixel values for 10,000 images of size 3*32*32. The first 1024 values in each row represent the red channel, the next 1024 represent the green values, and the final 1024 represent the blue channels. The second element in the dictionary is 'labels', a list of 10000 labels 0-9 corresponding to the order of 'data'. The final file in the archive also contains a Python dictionary, this one with 'label_names', a 10-element list which gives meaningful names to the numeric labels in the labels array described above. For example, label_names[0] == "airplane", label_names[1] == "automobile", etc.

Bootstrapping, first introduced in 1979 [63], is a resampling algorithm. It is a computationally intensive algorithm with straightforward implementation and powerful implications in estimation. An important question is how many times the data should be resampled. Efron suggests that number of resamplings of the order of 100 is enough to get reasonable estimates [64]. [65] considers a resampling of 500 very large. In practice today, many researchers use 5000, 10,000 or more because computational capacities easily allow for it. In our case, we have selected 150 iterations. We are training a mature neural network at each resampling, and performing this thousands of times is not practical or recommended.

Bootstrapping the neural network included training 150 unique learners for each of the architectural designs considered. There is no shared information between the learners, and thus the computational cost scales linearly with the number of learners. However, the independence of the models allowed for them to be trained in parallel as much as the computational resources allowed. The models were all trained on the Gilbreth cluster at Purdue University. Gilbreth is a community cluster optimized for communities running GPU intensive applications such as machine learning. It has 66 Dell compute nodes with Intel Xeon processors and 136 Nvidia Tesla GPUs. Gilbreth is named in honor of Lillian Moller Gilbreth, Purdue's first female engineering professor. The advanced computing resources are maintained and operated by Information Technology at Purdue (ITaP).

3.3.2 Bootstrap 3-Layer Convolutional Neural Network (CNN3)

Three networks were bootstrapped, on a total of two image datasets. Network architectures were selected for different datasets based on what is already known to work well for the datasets. The first network is a three layer convolutional neural network, which we will call CNN3. This model takes as input 2-dimensional, n x n black and white images. The input goes through a 2d convolution, followed by a rectified linear unit activation function and a max pooling. This is repeated three times. After these three convolutional layers, there is a dropout layer and finally a fully connected layer. The output of the fully connected layer is that of a linear transformation, with each image being assigned a value for each of the classes. The image will be classified as whichever class has the maximum value.

The CNN3 network was used to classify the MNIST dataset, described above. The images were preprocessed prior to training. Each image was transformed to a tensor image and normalized. Transformation to a tensor image takes the 28*28 = 784 input integer values within the range [0, 255] and outputs a 3-dimensional tensor of size (1, 28, 28) with values in the range [0, 1). Normalization then occurs by subtracting the mean and dividing by the standard deviation, 0.1307 and 0.3081, respectively. With an image input size of only 28 by 28 pixels, it was determined that each of the convolutions in the network would performed using the relatively small kernel size of 3, a stride of 1, and padding was set to 2. The max pooling was performed with a kernel size of 2 and a stride of 2. The dropout was performed during training at a rate of 0.5.

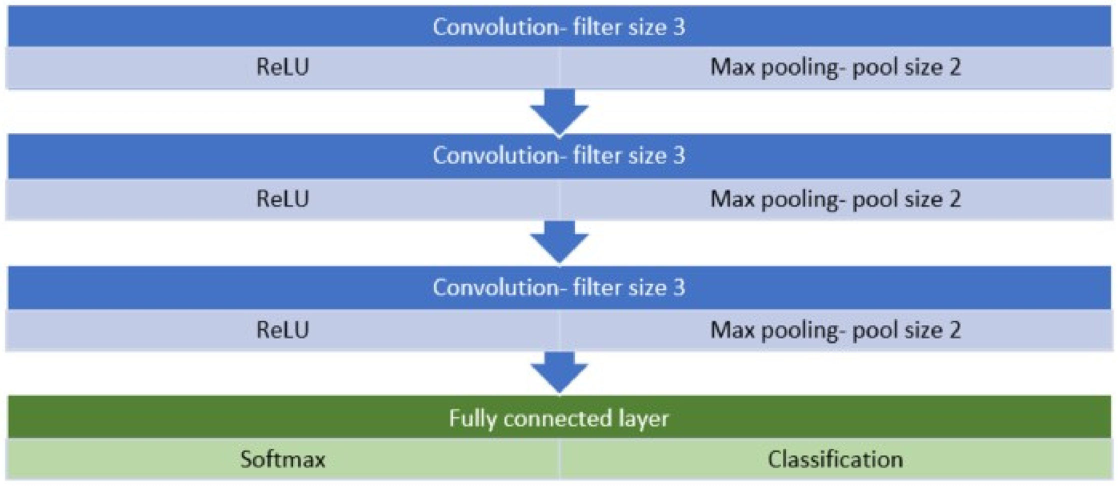


Figure 3.1. Model structure for CNN3

Training was performed by iterating through the training dataset 10 times. The data was randomly ordered during training, with a different randomization for each of the 10 epochs. Network parameters were initialized according to the Kaiming Uniform initialization. The data went through the model in batches of size 100. Optimization was done via the Adam algorithm, with a learning rate of 0.001. This same training technique was performed 150 times using 150 distinct training datasets selected via bootstrap sampling. 60,000 images were selected with replacement from the available 60,000 images. All parameter values were stored for each of the 150 networks.

Distribution of node values were visualized as both a boxplot and a histogram with a density curve. There were 320 parameters for the first convolutional layer, 18,496 for the second, 36,928 for the third, and 16,010 for the fully connected layer, for a total of 71,754 trainable parameters. The distribution of a selected number of the 71,754 parameters can be seen below. The plots are seen to be unimodal, therefore the sample size of 150 seems sufficient to use the results for parameter estimation.

3.3.3 Bootstrap VGG16

The second network is the VGG16 network proposed by Simonyan and Zisserman [66]. This network is comprised of 13 convolution layers and 3 fully connected layers, followed by a softmax layer. The 13 convolutional layers, and their corresponding activation functions, are separated into 5 blocks. Each block is followed by a max pooling layer. Following the five blocks, an adaptive average pooling is performed, followed by the fully connected block. The fully connected block proceeds with two repetitions of: dropout layer, linear function, activation function. A final linear transformation is applied with softmax, resulting in one value for each of the classes, with all values summing to 1.

The VGG16 network is used to train the CIFAR10 image dataset, described above. The images were preprocessed prior to training. Random images were flipped horizontally at a rate of 50%. Each image was then padded with 4 pixels on every side and randomly cropped to back to 32x32. Each image was also transformed to a tensor image and normalized. Transformation to a tensor image takes the 3*32*32 = 3072 input integer values within the range [0, 255] and outputs a 3-dimensional tensor of size (3, 32, 32) with values in the range [0, 1). Normalization then occurs by subtracting the mean and dividing by the standard deviation, [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225], respectively.

The convolutions all use a 3 x 3 kernel size, as in the CNN3 above. This small kernel size was necessary in order to achieve this network depth on a sequential convolutional neural network [66]. The number of trainable parameters can be seen in the following chart. The first two convolutional layers have 1,792 and 36,928 trainable parameters. This value increases as the model progresses, with a maximum of 2,359,808 parameters seen in the last five convolutional layers. The first fully connected layer jumps to 12,845,568 trainable parameters, with the second containing 262,656, and the final fully connected layer including

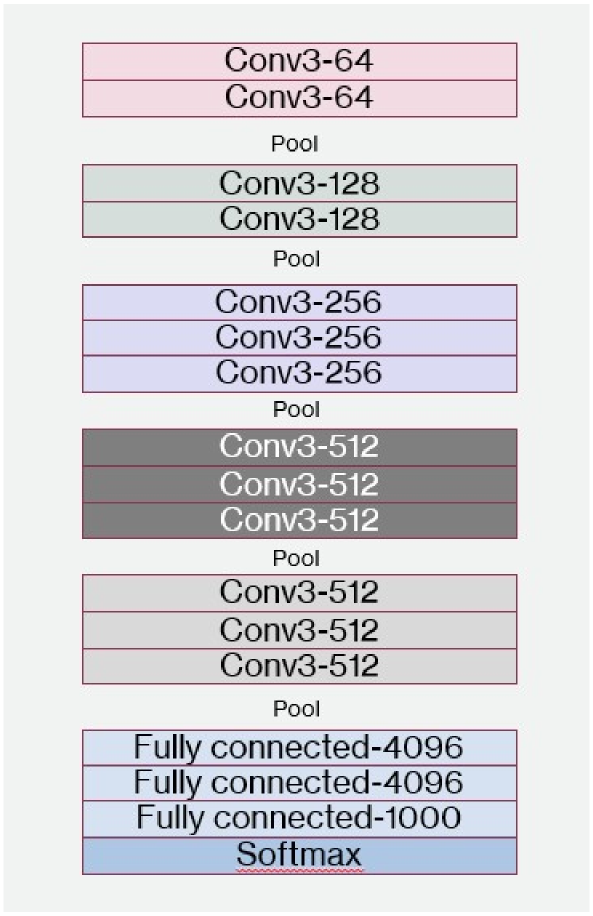


Figure 3.2. Model structure for VGG-16

5,130 trainable parameters. With a total of 27,828,042 trainable parameters, this is the slowest of the models considered in this work.

Prior to training on CIFAR-10, the weights were set to pre-trained values available in the Torchvision library. The weights were learned from the ImageNet database. PyTorch reports a Top-1 accuracy of 71.592% on the ImageNet dataset. However, ImageNet has 1000 classes as compared with CIFAR-10's ten. Thus, the three fully connected layers must be reset to the correct size to accommodate the correct output size. Resizing the linear layers also loses the pretrained weights. Weights are then initialized via the Keiming algorithm, described above. During training, all weights are updated, including both the pre-trained and newly initialized weights.

Training was performed by iterating through the training dataset 180 times. As with the training of the MNIST data on CNN3, the data was randomly assigned during training, with a different randomization for each of the 180 epochs. The data went through the model in batches of size 256. Optimization was done using stochastic gradient descent, with momentum 0.9, weight decay of 0.0005, and an initial learning rate of 0.05. The learning rate decayed by a factor of 0.1 every 5 epochs. The same hyperparameters were used to train 150 unique networks, each using a bootstrapped sampling of the full CIFAR10 training data as the training data for the particular instance of the network. 50,000 images were selected with replacement from the 50,000 CIFAR10 training image dataset. The 27,828,042 trainable parameters had their values stored for each of the 150 models.

The generation of plots for each of the 27.8 million parameters was not feasible. In this case, a random selection of plots were generated. Ten weight values and ten bias values were selected randomly without replacement from each layer. The plots of these randomly selected nodes were generated, and a selection of them are shown in Section 3.5.

3.3.4 Bootstrap Inception v3

The third network is the third iteration of the Inception framework Szegedy, Vanhoucke, Ioffe, *et al.* [67], so named for the 'network in network' set-up as well as a popular (at the time) meme based on the movie of the same name. Utilizing a 'network in network' framework, called Inception modules, Inception v3 contains 42 layers, making it the most complex of the models considered in this work.

The motivation for the inception modules is to have a sparsely connected architecture, even within the convolutional layers. The inception modules make heavy use of 1 x 1 convolution filters. These 1 x 1 layers are applied before a larger filter size. The authors included these layers as a means of dimension reduction, as well as to increase representational power of the network [67]. This allows for more layers with a fewer number of overall parameters. Inception v3 uses three Inception modules, which we will label as modules A, B, and C. Modules B and C are more complex than Module A and can be seen in the Figure below. Module A consists of four parallel 1 x 1 convolution layers. The first of which is then passed to two 3 x 3 filters sequentially. The second is passed to a single 3 x 3 filter, the third undergoes max pooling before the 1 x 1 layer, and the final one is just a single 1 x 1 layer. The results of these four parallel networks are then placed into a single output filter bank and become the input for the next module. Thus the subsequent modules are focusing on the local regions of interest from the previous module. The different layer types and filter sizes are designed to capture different parts of the input image. The 1 x 1 filters focus on the correlation in local regions. The larger filter sizes are designed to identify features of higher abstraction.

The first five layers are traditional convolution layers, each of which is followed by a batch normalization. The third and fifth are also followed by max pooling. After the first five convolution layers, there are three repetitions of Inception module A, five repetitions of Inception module B, and two repetitions of Inception module C. The output of these layers are input to an adaptive average pooling layer, following by a dropout layer with rate 0.4. and finally a fully connected linear transformation with softmax.

The Inception v3 architecture also makes use of an auxiliary classifier after the final round of Inception module B, near the end of the model. The auxiliary classifier, used only during training, acts as a regularizer and helps with convergence in late training [67]. During training, the total loss is computed as loss1 + 0.4*loss2, where loss1 is the loss associated with the full network, and loss2 is the loss associated with the auxiliary branch.

PyTorch's Inception v3 network has the constraint that the input image size must be 299 x 299 color images. This network was used to learn the CIFAR-10 dataset, with image size of 32x32 color images. In order to use this dataset, it was necessary to first increase the image size. This was performed by first resizing the images to 325x325 by padding the images on all sides, then taking the center crop of the required 299x299. The resultant images are surrounded by large black borders. Each image was also transformed to a tensor image and normalized. Transformation to a tensor image takes the 3*299*299 = 268,203 input integer values within the range [0, 255] and outputs a 3-dimensional tensor of size (3, 299, 299) with values in the range [0, 1). Normalization then occurs by subtracting the mean and dividing by the standard deviation, [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225], respectively.

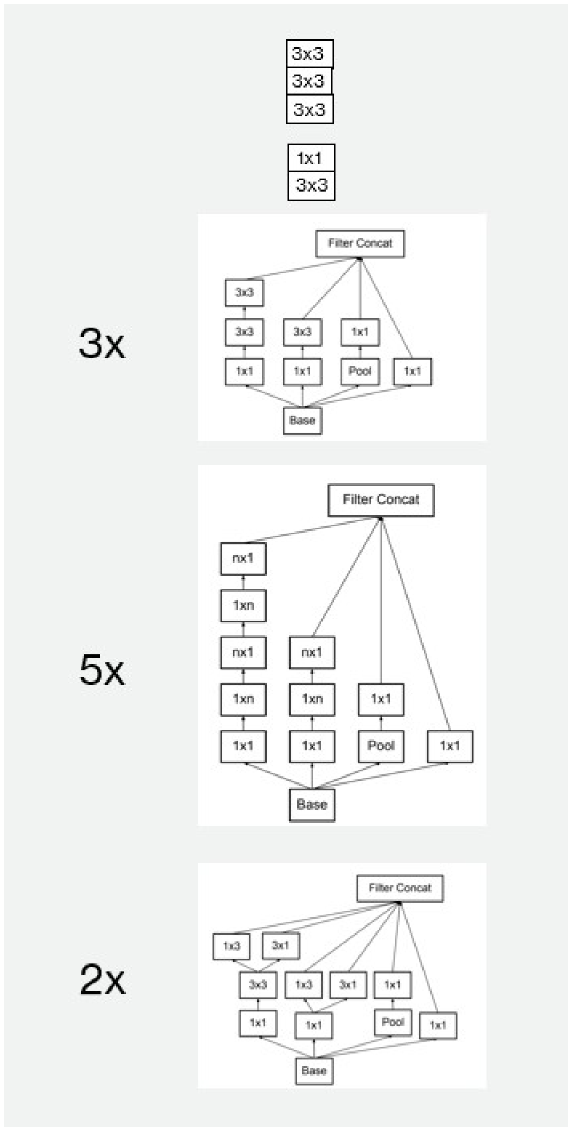


Figure 3.3. Model structure for Inception v3

The strategic filtering technique keep the number of parameters in this network relatively small as compared the VGG framework. Although there are over 2.5 times the number of layers, Inception v3 has 21,806,058 trainable parameters as compared with over 27.8 million for VGG16. Similarly, the computational efficiency is superior as compared with the more straightforward VGG framework. There is the additional benefit that the number of parameters will be unchanged for any image size up to 299x299 in the Inception v3 network.

Meanwhile, the number of trainable parameters for VGG16 would jump to over 138 million for an image of this size.

Prior to training on CIFAR-10, the weights were set to pre-trained values available in the Torchvision library. The weights were ported in directly from the Inception v3 paper. PyTorch reports a Top-1 accuracy of 77.294% on the ImageNet dataset. As above, the number of parameters in the fully connected layer is incorrect because the number of classes differs between the pretrained dataset and our dataset. After creating linear transformation layers of the correct dimension, weights are initialized via the Keiming algorithm, described above. During training, all weights are updated, including both the pre-trained and newly initialized weights.

Training was performed by iterating through the training dataset 180 times. As with the training of the previously mentioned networks, the data was randomly assigned during training, with a different randomization for each of the 180 epochs. The data went through the model in batches of size 64. Optimization was done using stochastic gradient descent, with momentum 0.9, weight decay of 0.0005, and an initial learning rate of 0.05. The learning rate decayed by a factor of 0.1 every 5 epochs. The same hyperparameters were used to train 150 unique networks, each using a bootstrapped sampling of the full CIFAR10 training data as the training data for the particular instance of the network. 50,000 images were selected with replacement from the 50,000 CIFAR10 training image dataset. The values of each of the 21,806,058 trainable parameters were stored for each of the 150 models.

As with the VGG16 results, only a random selection of plots were generated. As above, ten weight values and ten bias values were selected randomly without replacement from each layer. The plots of these randomly selected nodes were generated, and a selection of them are shown in Section 3.5.

3.4 Models Generated Using Full Training Dataset by Varying Random Initial Seeds

In addition to bootstrapped neural networks, networks were also trained using the full training data. The network structure is fixed and multiple copies of the models are trained by varying the random initial seeds. This shows the stochastic nature of the non-convex

Table 3.1. Experiment setup summary for bootstrap.

	Dataset	**Models**	**Layers**	**trainable param**	**Training epochs**	**Batch size**
CNN3	MNIST	150	4	71,754	10	100
VGG16	CIFAR-10	150	16	27,828,042	180	256
Incept v3	CIFAR-10	150	42	21,806,058	180	64

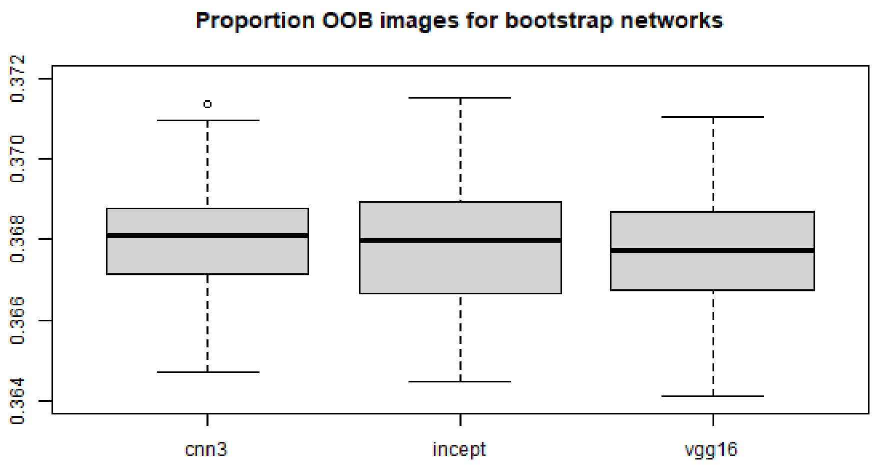


Figure 3.4. Proportion OOB images for bootstrap networks.

optimization training process. As with the bootstrapped networks, 150 models were generated for each network type. These included CNN3 trained on MNIST and VGG16 trained on CIFAR-10, as before, as well as ResNet-18 traind on CIFAR-10. The CNN3 and VGG16 networks follow the same strategies as outlined in sections 3.3.2 and 3.3.3, with one noted exception. Rather than selecting bootstrapped samples of the training data, the entire training set is used in each of the 150 fully trained networks. The data is still going through the model in a random order with different initial seeds, but the model is seeing the full training set in every network.

The third network, ResNet-18 [68] was developed to address a problem of degradation identified in other 'very deep' models (those with 16 or more layers), such as VGG. The excessive depth of the models were observed to degrade the accuracy. Such degradation was not due to overfitting, because the training errors were also increasing [68]. If layers of identity mapping are injected into a shallower model to create a deeper model which would have no lower training accuracy than the shallow model. In practice, the ResNet architecture injects layers which fit a residual mapping. Each residual learning building block is essentially an identity mapping, and it has been observed that models composed of such blocks can have over 100 layers without degradation of training accuracy.

In this work, we consider the 18 layer architecture, ResNet-18. The model begins with a convolutional layer with 7x7 filter size, stride of 2 and padding of 3. This layer outputs a mapping with channel depth 64. After batch normalization, a ReLU activation layer and max pooling, a series of eight residual building blocks are trained. Each residual building block is comprised of two convolutional layers. The convolutional layers increase from channel depth 64 in the first two blocks to 128 in the next two, 256 in the next two, and 512 in the final two. A downsampling is performed each time the channel depth increases. Each of the convolutional layers in the residual building blocks have kernel size 3x3, with padding and stride of 1. Finally, there is an adaptive pooling layer and a fully connected layer. The network structure and residual building block can be visualized in Figure 3.5.

The ResNet-18 model was trained on the CIFAR-10 dataset. Using the 3x32x32 image size results in a total of 11,181,642 total trainable parameters. This is approximately 2.5 times fewer parameters than the similarly deep VGG16 model. The accuracy on clean CIFAR-10 images is also found to be superior on our re-trained ResNet-18 model as compared with our re-trained VGG16 model. This was done using the ResNet-18 model available from Python's Torchvision library. This model has been pre-trained on ImageNet and PyTorch reports a Top-1 accuray of 69.758 on ImageNet. After correcting the linear output layer dimension to accommodate the class size of 10, weights are initialized via the Kaiming Uniform algorithm.

This pretrained model was trained on CIFAR-10 using similar methods as with retraining VGG16 (section 3.3.3) and Inception v3 (section 3.3.4). The images were first preprocessed with random horizontal flipping, padding with random cropping, transformation to tensor and normalization, as with other networks trained on CIFAR10. The full database of 50,000 training images was used in retraining. There were 300 epochs of training, and a batch size

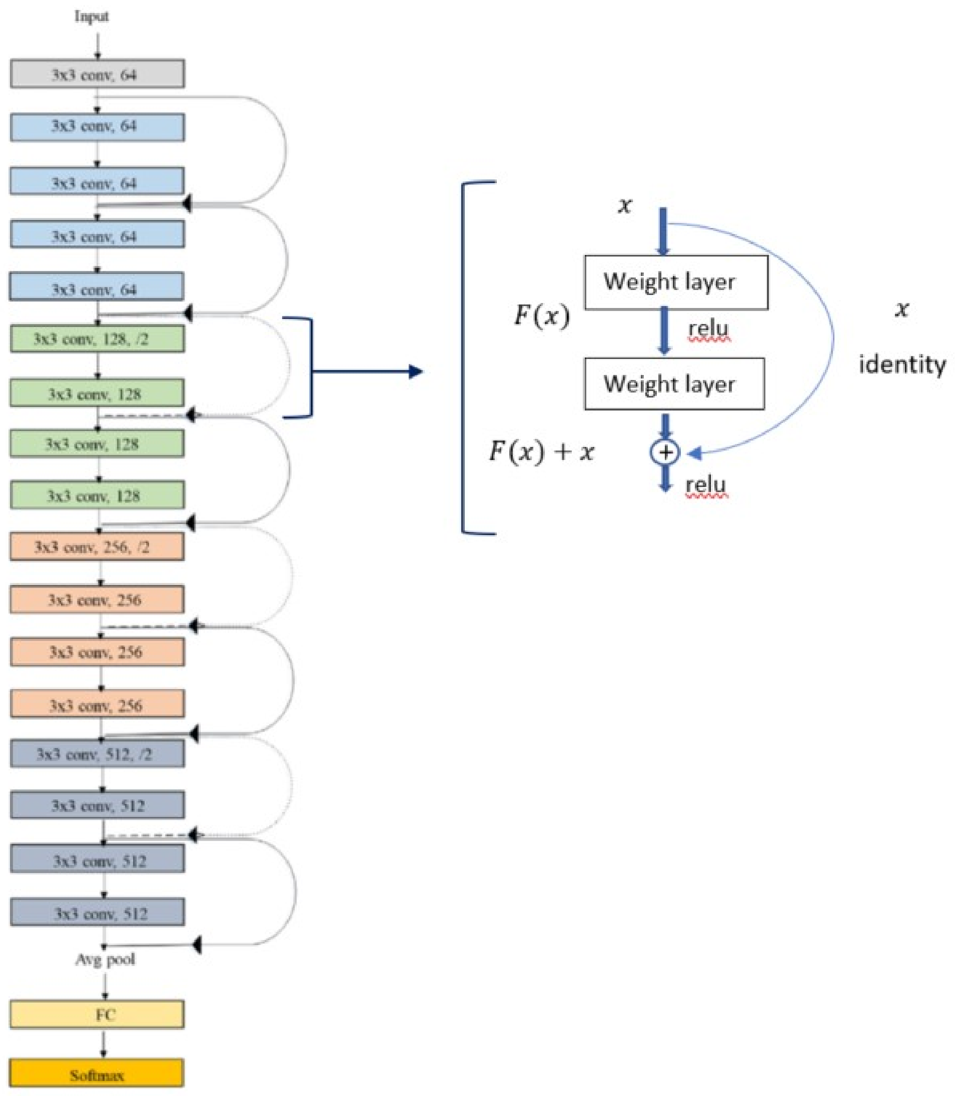
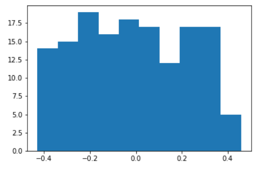


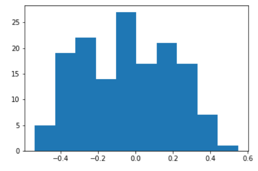
Figure 3.5. Model structure for ResNet-18

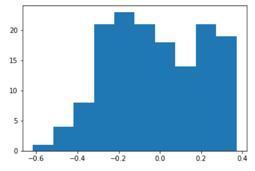
of 256. Optimization was performed with SGD, with momentum of 0.9 and weight decay 0.0005. The initial learning rate of 0.05 which decayed by a factor of 0.5 every 30 epochs. This process was repeated 150 times.

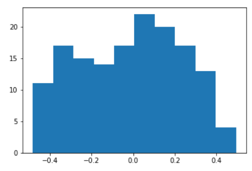
Table 3.2. Experiment setup summary for random ordering of full training data.

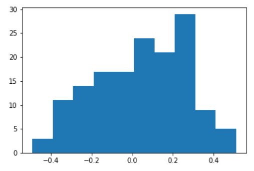
	Dataset	Models	Layers	trainable param	Training epochs	Batch size
CNN3	MNIST	150	4	71,754	5	100
VGG16	CIFAR-10	150	16	27,828,042	300	256
ResNet18	CIFAR-10	150	18	11,181,642	300	256


(a)


(b)


(c)


(d)


(e)

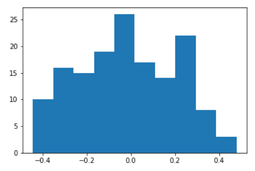

(f)

Figure 3.6. Distribution of weights selected from bootstrap CNN3, first convolutional layer.

3.5 Bootstrapped CNN Model Parameter Distribution

The bootstrap weight distributions of the first convolutional layer for the CNN3 model trained on MNIST data are approximately symmetric, as shown in Figure 3.6 with selected ones. The bootstrap bias distributions of the first convolutional layer are approximately symmetric too, as shown in Figure 3.7. Both the weights and the biases are in the +/- 0.4 range.

The bootstrap weight distributions of the fully connected layer for the CNN3 model trained on MNIST data are approximately symmetric, as shown in Figure 3.8 with selected ones. The bootstrap bias distributions of the fully connected layer are approximately sym-

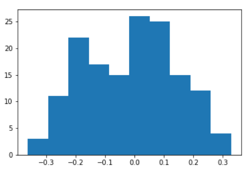
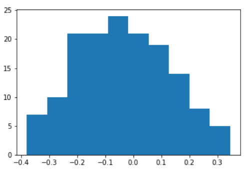
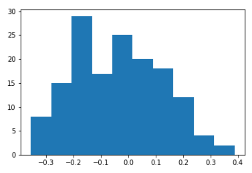
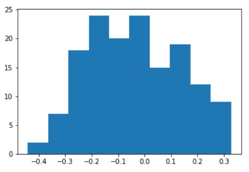
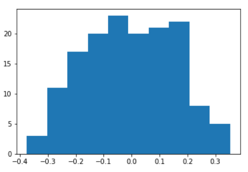
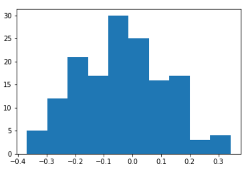

(a)

(b)

(c)

(d)

(e)

(f)

Figure 3.7. Distribution of biases selected from bootstrap CNN3, first convolutional layer.

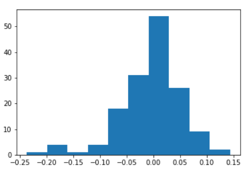
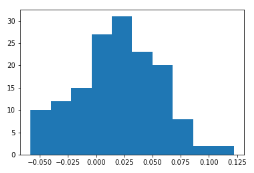
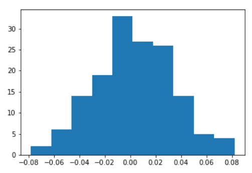
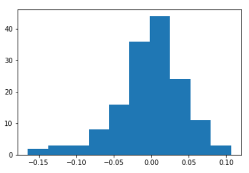
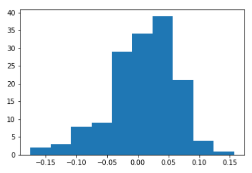
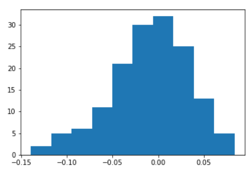

(a)

(b)

(c)

(d)

(e)

(f)

Figure 3.8. Distribution of weights selected from bootstrap CNN3, fully connected layer.

metric too, as shown in Figure 3.9. Weights are primarily in in the +/- 0.15 range, while the biases are in the +/- 0.05 range.

The bootstrap weight distributions of the first convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.10 with selected ones. Heavy tails are also observed in a few of the selected examples, as in (b), (c),

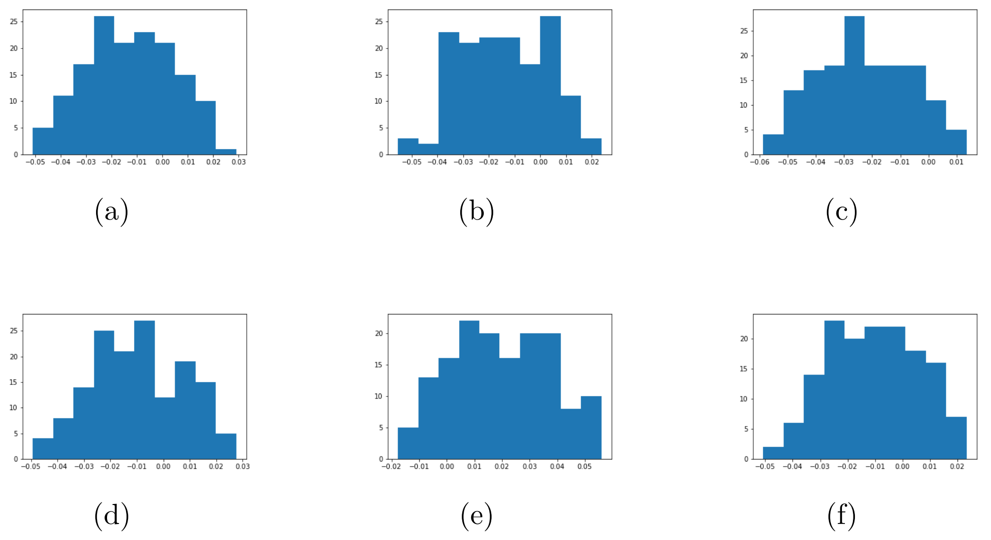


Figure 3.9. Distribution of biases selected from bootstrap CNN3, fully connected layer.

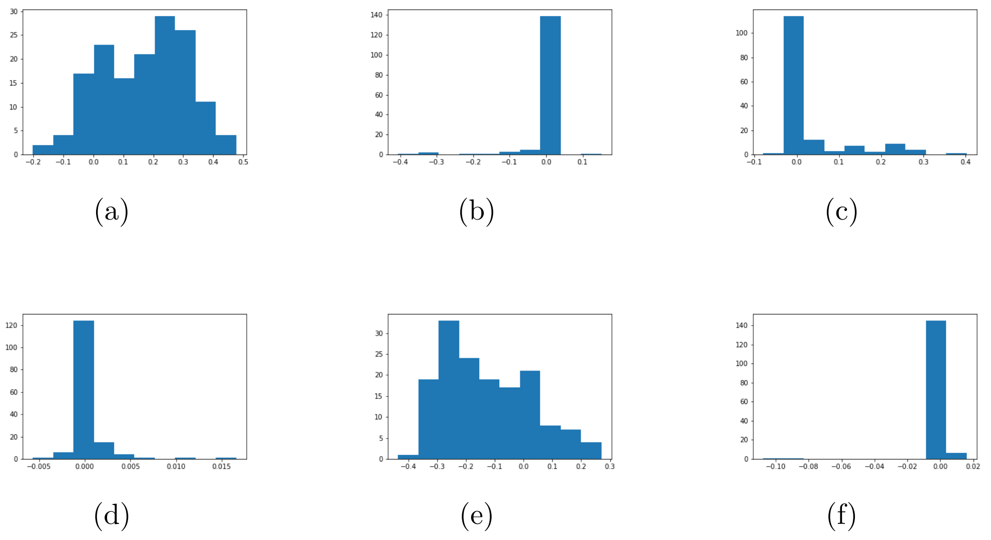


Figure 3.10. Distribution of weights selected from bootstrap Inception v3, first convolutional layer.

(d) and (f). Weights without heavy tails are approximately in the +/- 0.4 range, while the ranges of the heavy-tailed weights are sometimes smaller.

The bootstrap weight distributions of the second convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.11 with selected ones. Heavy tails are also observed in the selected figures. Weights are approximately 0 with most ranges less than +/- 0.01.

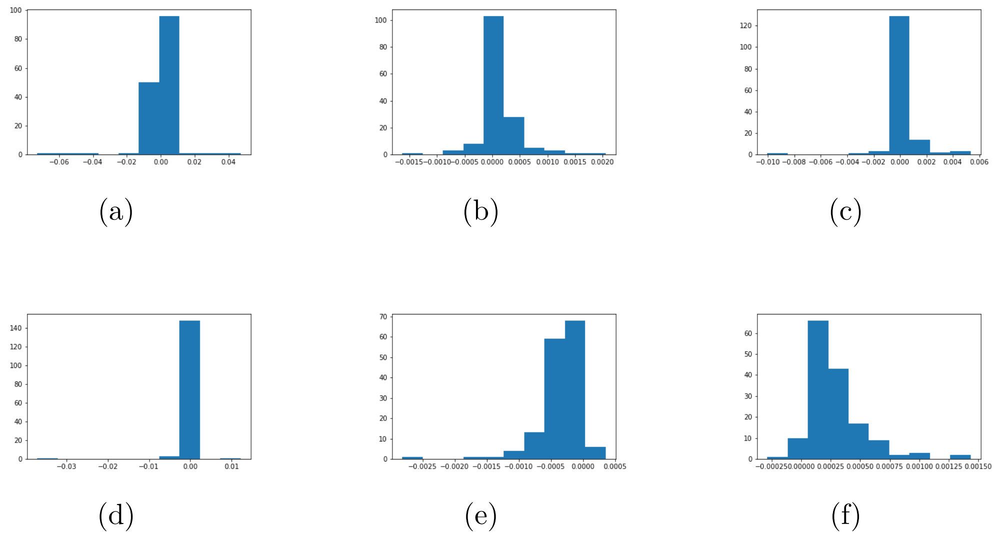


Figure 3.11. Distribution of weights selected from bootstrap Inception v3, second convolutional layer.

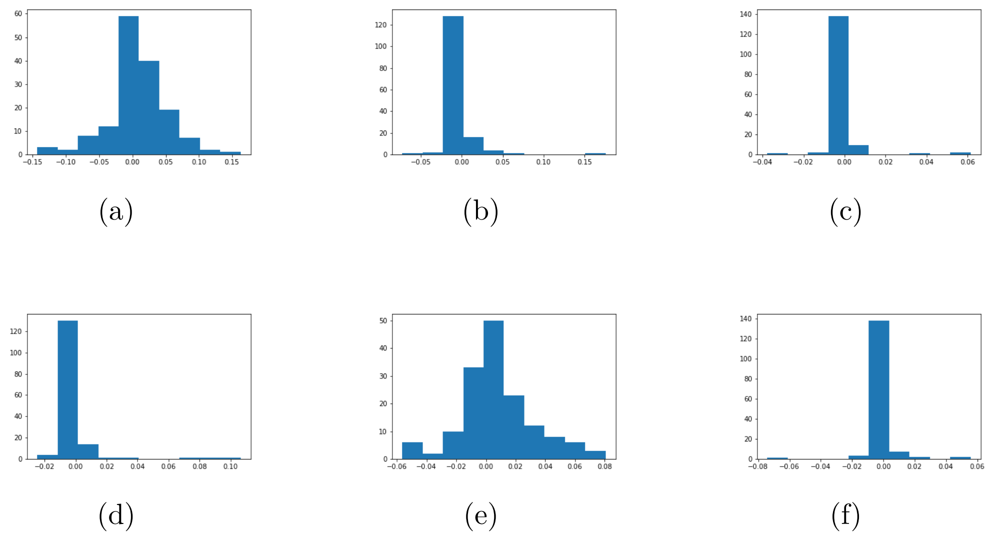


Figure 3.12. Distribution of weights selected from bootstrap Inception v3, third convolutional layer.

The bootstrap weight distributions of the third convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.12 with selected ones. Heavy tails are observed in several weights, as seen in sub-figures (b), (c), (d), and (f). Weights are approximately in the +/- 0.15 range.

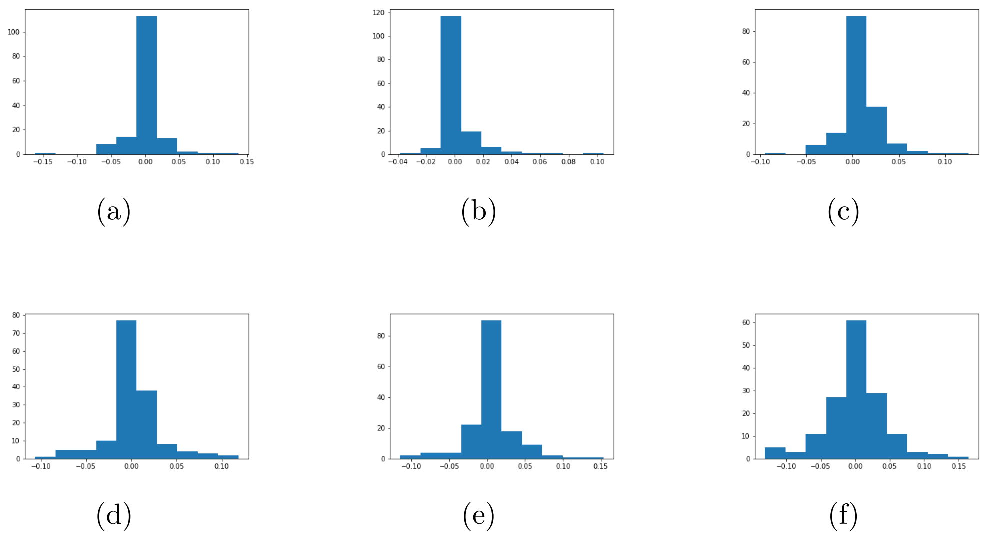


Figure 3.13. Distribution of weights selected from bootstrap Inception v3, fourth convolutional layer.

The bootstrap weight distributions of the fourth convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.13 with selected ones. Weights are approximately in the +/- 0.15 range.

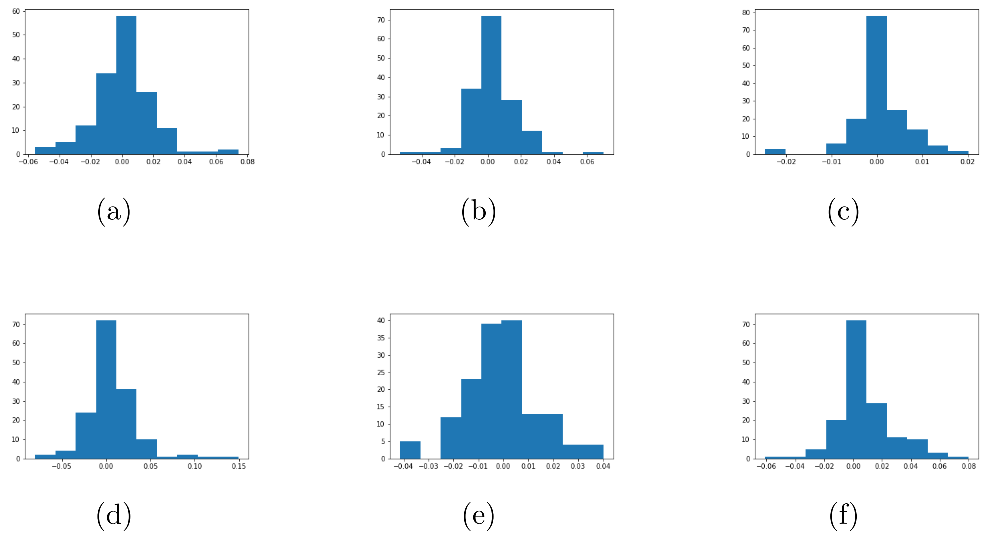


Figure 3.14. Distribution of weights selected from bootstrap Inception v3, fifth convolutional layer.

The bootstrap weight distributions of the fifth convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.14 with selected ones. Weights are approximately in the +/- 0.05 range, with the exception of some heavy tails.

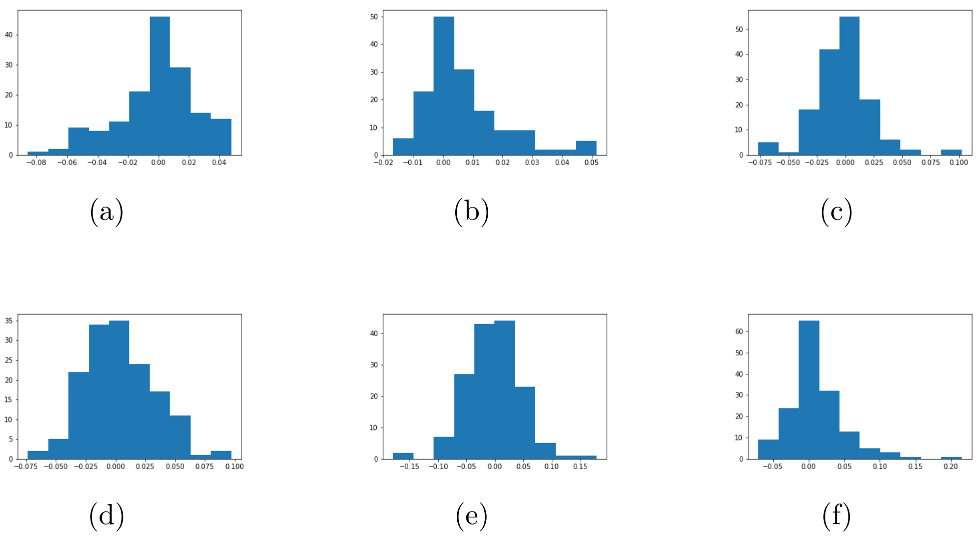


Figure 3.15. Distribution of weights selected from bootstrap Inception v3, sixth convolutional layer.

The bootstrap weight distributions of the sixth convolutional layer for the Inception v3 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.15 with selected ones. Weights are approximately in the +/- 0.1 range, with the exception of some heavy tails.

The bootstrap weight distributions of the first convolutional layer for the VGG16 model trained on CIFAR-10 data are approximately symmetric, as shown in Figure 3.16 with selected ones. The bootstrap bias distributions of the first convolutional layer are approximately symmetric too, as shown in Figure 3.17. Both the weights and the biases are in the +/- 0.1 range.

The bootstrap weight distributions of the final fully connected layer for the VGG16 model trained on CIFAR-10 data are approximately symmetric, as shown in Figure 3.18 with selected ones. The bootstrap bias distributions of the final fully connected layer are approximately symmetric too, as shown in Figure 3.19. Both the weights and the biases are in the +/- 0.15 range, with the exception of some heavy tails.

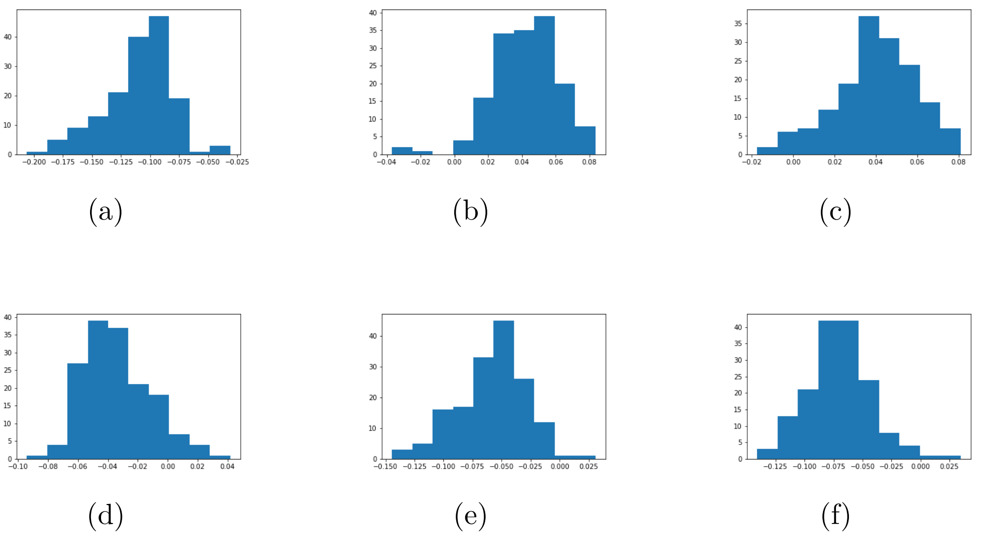


Figure 3.16. Distribution of weights selected from bootstrap VGG-16, first convolutional layer.

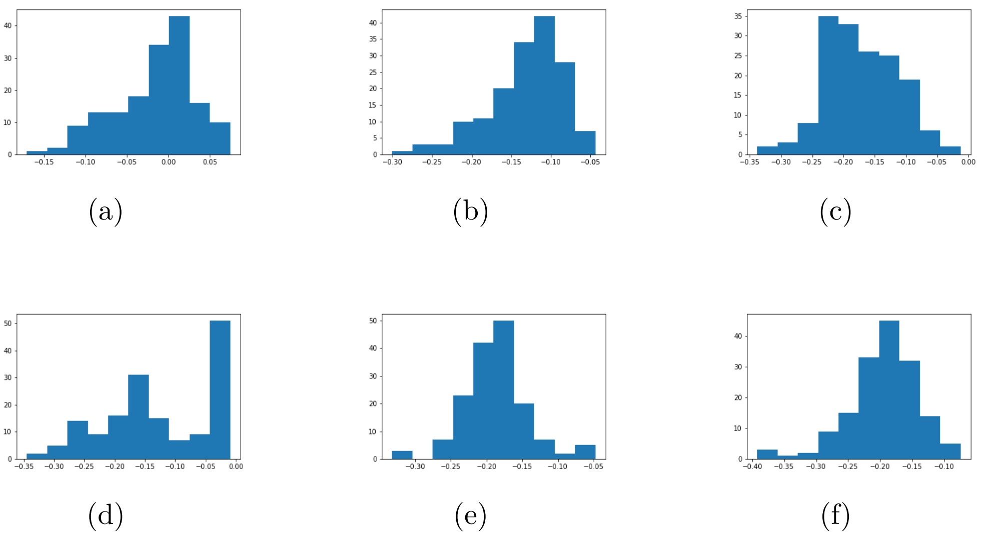


Figure 3.17. Distribution of biases selected from bootstrap VGG-16, first convolutional layer.

A goodness of fit test must be performed to determine whether the assumption that individual weights are distributed according to a normal distribution is appropriate. The Kolmogorov-Smirnov test [69] for goodness of fit (KS test) was presented in 1951, expanding on the work of Andrey Kolmogorov and Nikolai Smirnov. The test statistic is a quantification of the distance between the empirical distribution of a sample and the cumulative distribution

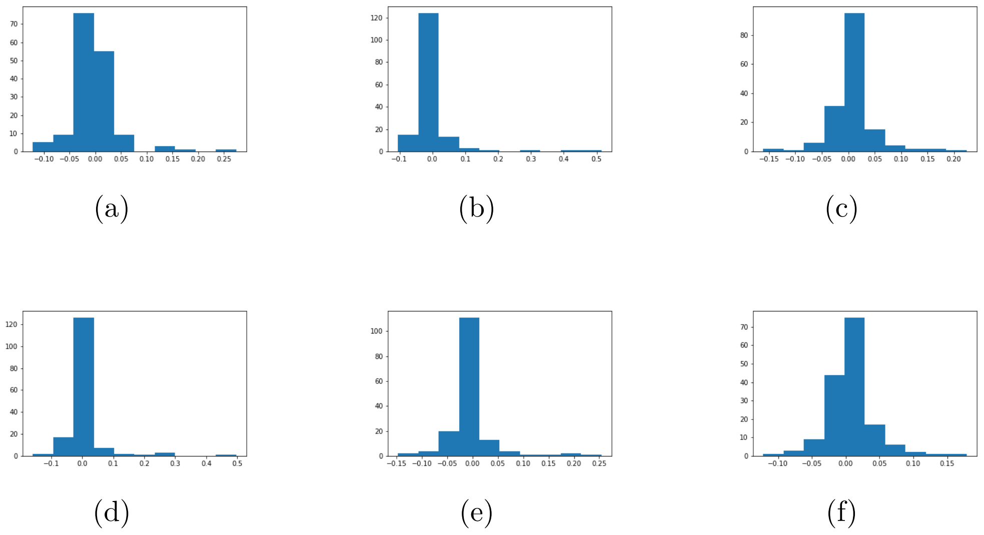


Figure 3.18. Distribution of weights selected from bootstrap VGG-16, final fully connected layer.

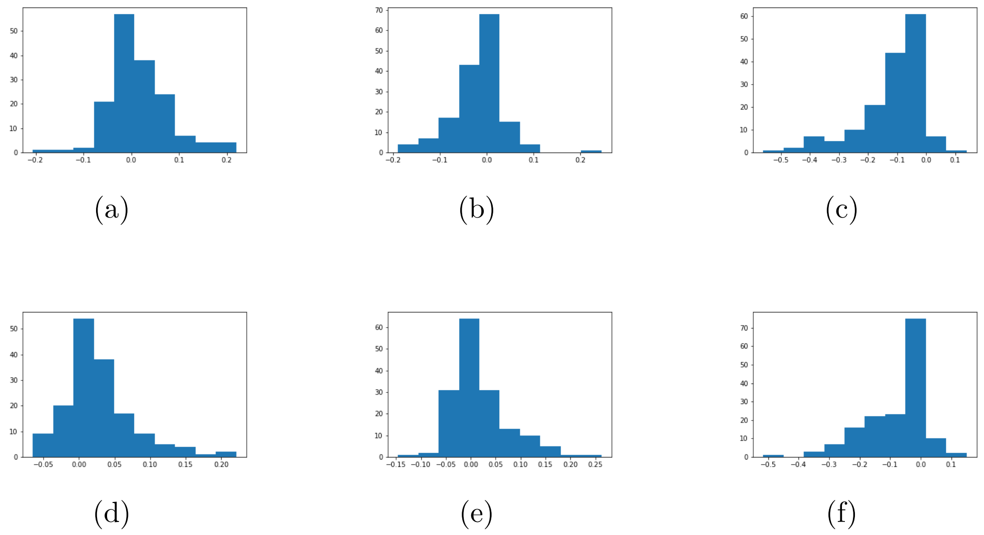


Figure 3.19. Distribution of biases selected from bootstrap VGG-16, final fully connected layer.

of a comparison distribution. The test statistic is the largest observed difference between the empirical cumulative distribution and the reference cumulative distribution. KS test statistics are therefore bounded between 0 and 1. Two identical distributions will observe a value of 0. In this section and 3.6, the empirical distribution of individual weights will be compared to a univariate Gaussian. The KS test can also be used to measure the distance between two empirical samples, as in Section 3.7.

The null hypothesis of the test is that the empirical sample is drawn from the same distribution as the comparison distribution. In the two sample case, the null hypothesis is that the two samples are drawn from the same distribution. A cutoff of largest allowable distance is defined for a given confidence level. If the largest distance exceeds that value, the null hypothesis is rejected. Null hypotheses may also be defined as one-way tests, but in our work, all tests will be two-way, i.e., non-directional equality between the distributions.

In the one sample case with sample size n, the Kolmogorov-Smirnov statistic, D_n, is computed as

$$D_n = \sup_x |F_n(x) - F(x)| \tag{3.1}$$

where $F(x)$ is the cumulative distribution function of the reference distribution, and $F_n(x)$ is the empirical cumulative distribution of the sample. In this work, the reference distribution is a standard univariate Gaussian. The empirical cumulative distribution is defined as

$$F_n(x) = \frac{\text{number of elements in the sample} \leq x}{x} = \frac{1}{n}\sum_{\mathrm{i}-1}^{n} 1_{(-\infty,x]}(X_\mathrm{i}) \tag{3.2}$$

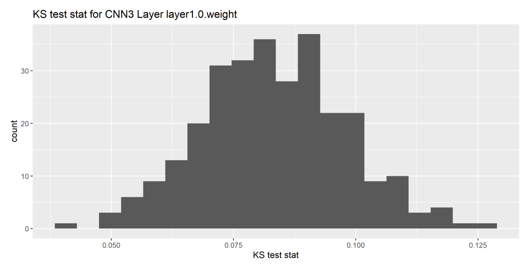
where $1_{(-\infty,x]}(X_\mathrm{i})$ is the indicator function, equal to 1 if $X_\mathrm{i} \leq x$, otherwise equal to 0.

The KS test is a goodness of fit test which can be constructed from the Kolmogorov distribution, K, as described in [69]. The distribution of K is defined as the supremum of a Brownian bridge. This test rejects the null hypothesis that the sample comes from the reference distribution at level α if

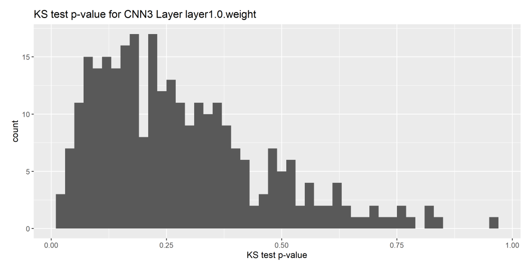
$$\sqrt{n}D_n > K_\alpha \tag{3.3}$$

where K_α is defined as the value at which the cumulative distribution of K is equal to $1 - \alpha$. All tests in this work are evaluated at level $\alpha = 0.05$.

For all bootstrapped networks, the KS test statistic was computed for up to 1000 weights and biases in all layers. Any layer with fewer than 1000 weights (or 1000 biases) has all weights (or biases) included in the following tables and plots. For layers with more than 1000 weights or biases, a random sampling of 1000 weights and biases was taken.



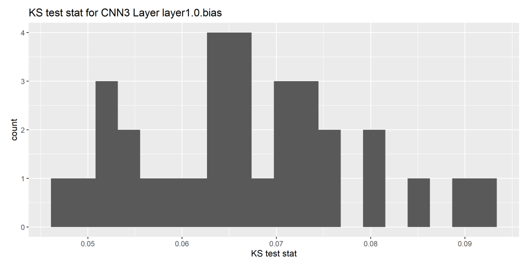
(a) KS test statistic



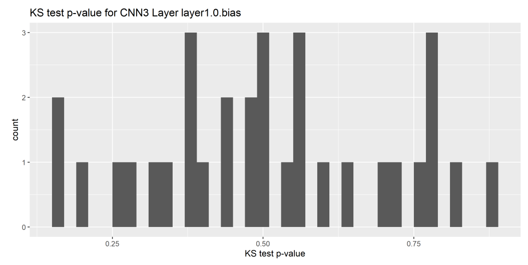
(b) KS test p-value

Figure 3.20. KS test results for selected weights in bootstrap CNN3, first convolutional layer.

Figure 3.20 shows that over 95% of the bootstrap weight distributions of the first convolutional layer of CNN3 are normal as confirmed by the KS test, with a p-value threshold of 0.05.



(a) KS test statistic

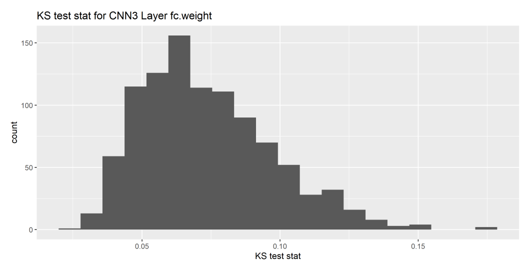


(b) KS test p-value

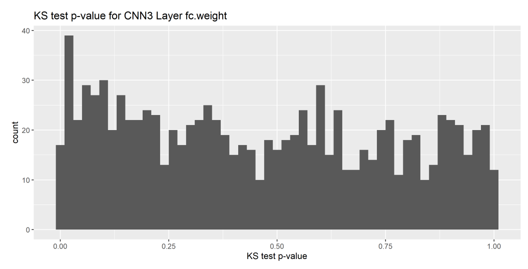
Figure 3.21. KS test results for selected biases in bootstrap CNN3, first convolutional layer.

Figure 3.21 shows that over 99% of the bootstrap bias distributions of the first convolutional layer of CNN3 are normal as confirmed by the KS test, with a p-value threshold of 0.05.

Figure 3.22 shows that over 90% of the bootstrap weight distributions of the fully connected layer of CNN3 are normal as confirmed by the KS test, with a p-value threshold of 0.05.

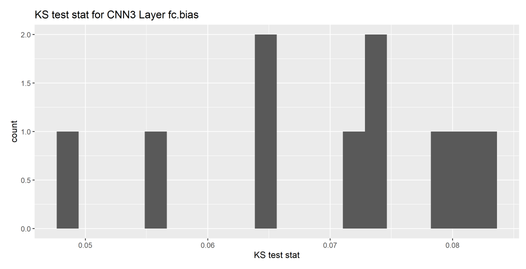


(a) KS test statistic

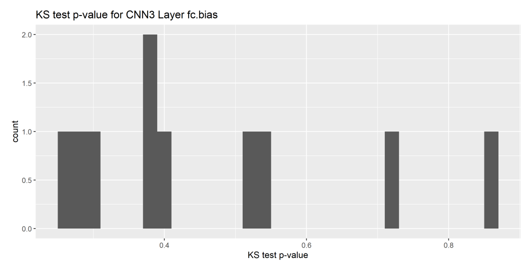


(b) KS test p-value

Figure 3.22. KS test results for selected weights in bootstrap CNN3, fully connected layer.



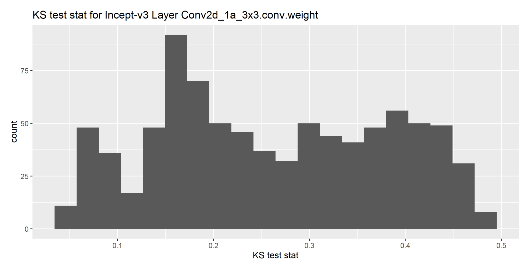
(a) KS test statistic



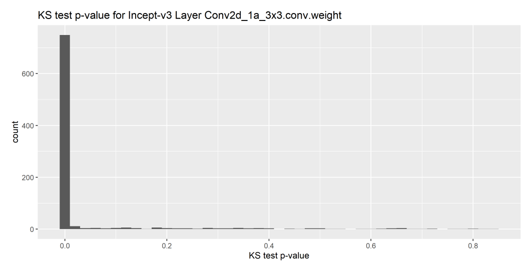
(b) KS test p-value

Figure 3.23. KS test results for selected biases in bootstrap CNN3, fully connected layer.

Figure 3.23 shows that all 10 of the bootstrap bias distributions of the fully connected layer of CNN3 are normal as confirmed by the KS test, with a p-value threshold of 0.05.



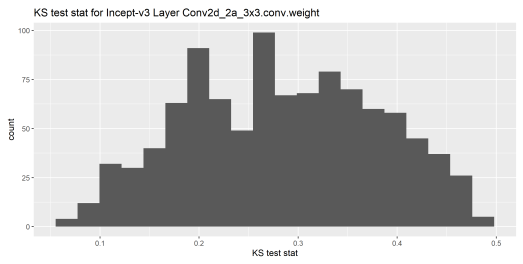
(a) KS test statistic



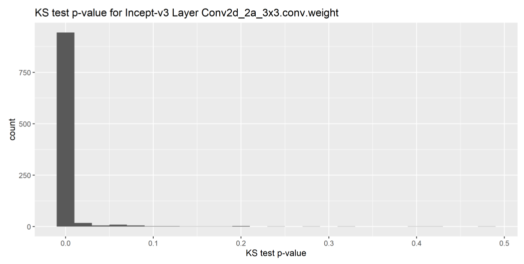
(b) KS test p-value

Figure 3.24. KS test results for selected weights in bootstrap Inception v3, first convolutional layer.

Figure 3.24 shows that over 75% of the bootstrap weight distributions of the first convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



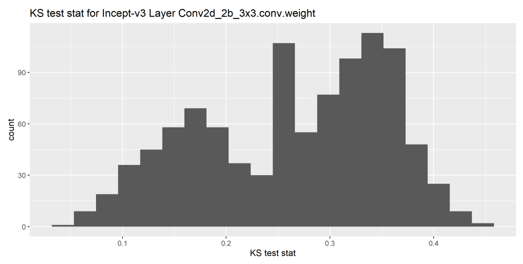
(a) KS test statistic



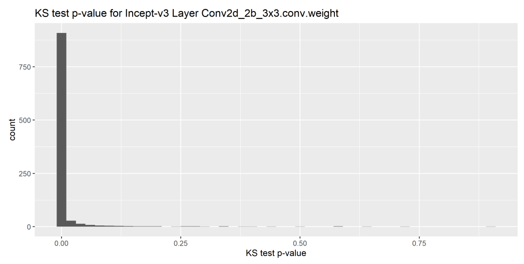
(b) KS test p-value

Figure 3.25. KS test results for selected weights in bootstrap Inception v3, second convolutional layer.

Figure 3.25 shows that over 95% of the bootstrap weight distributions of the second convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



(a) KS test statistic



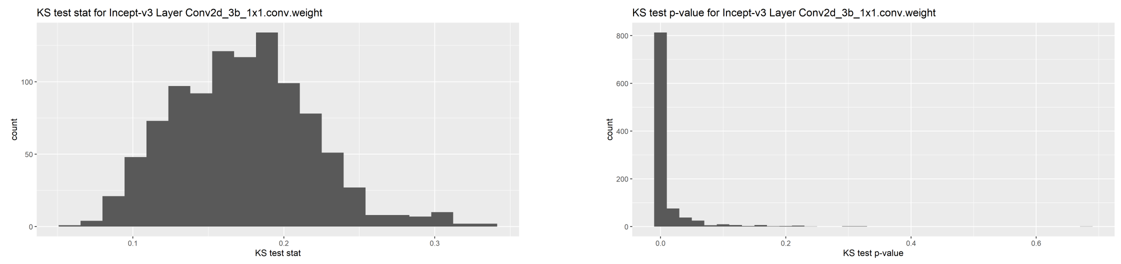
(b) KS test p-value

Figure 3.26. KS test results for selected weights in bootstrap Inception v3, third convolutional layer.

Figure 3.26 shows that over 95% of the bootstrap weight distributions of the third convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.

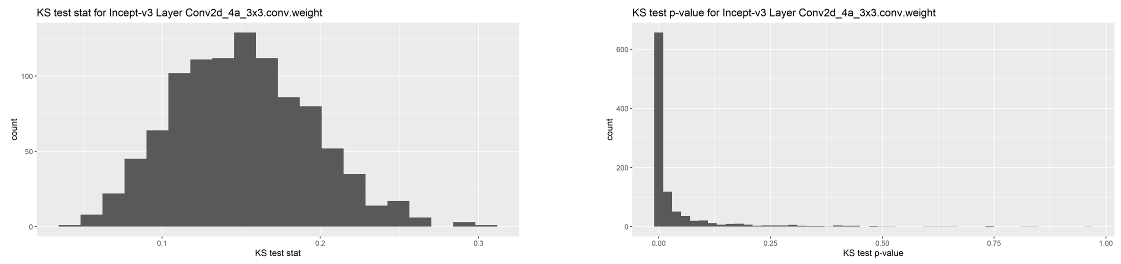
Figure 3.27 shows that over 90% of the bootstrap weight distributions of the fourth convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.

Figure 3.28 shows that over 75% of the bootstrap weight distributions of the fifth convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



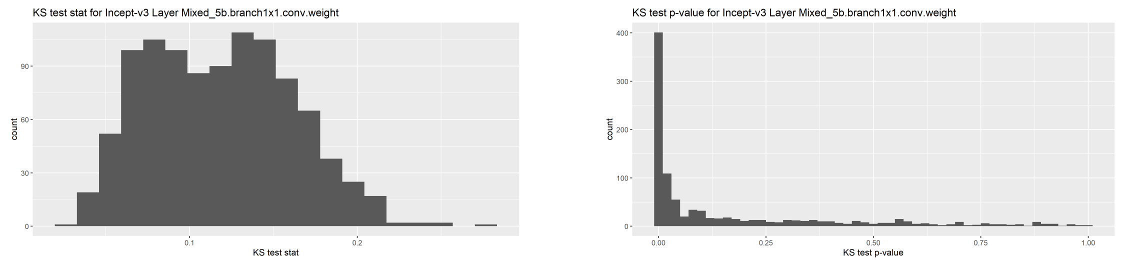
(a) KS test statistic (b) KS test p-value

Figure 3.27. KS test results for selected weights in bootstrap Inception v3, fourth convolutional layer.



(a) KS test statistic (b) KS test p-value

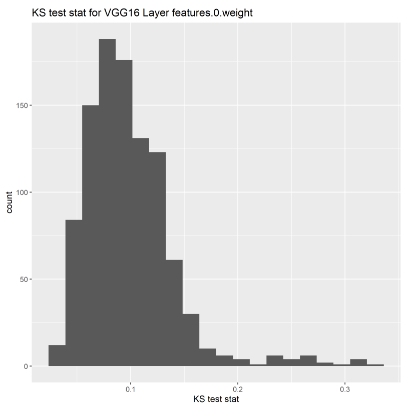
Figure 3.28. KS test results for selected weights in bootstrap Inception v3, fifth convolutional layer.



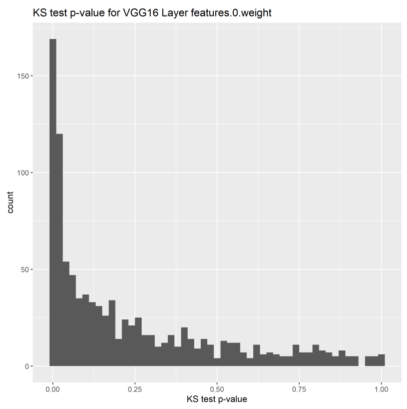
(a) KS test statistic (b) KS test p-value

Figure 3.29. KS test results for selected weights in bootstrap Inception v3, sixth convolutional layer.

Figure 3.29 shows that over 50% of the bootstrap weight distributions of the sixth convolutional layer of Inception v3 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



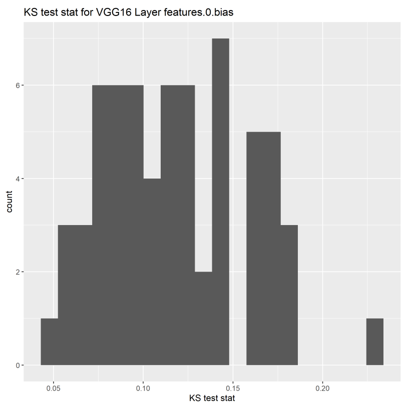
(a) KS test statistic



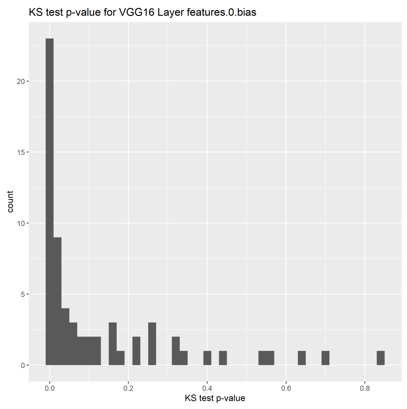
(b) KS test p-value

Figure 3.30. KS test results for selected weights in bootstrap VGG-16, first convolutional layer.

Figure 3.30 shows that approximately half of the bootstrap weight distributions of the first convolutional layer of VGG-16 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



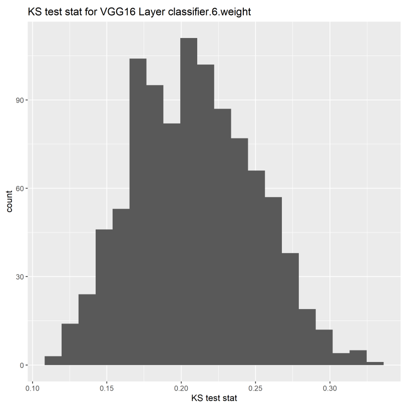
(a) KS test statistic



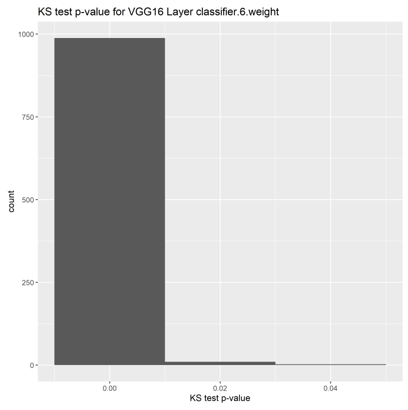
(b) KS test p-value

Figure 3.31. KS test results for selected biases in bootstrap VGG-16, first convolutional layer.

Figure 3.31 shows that over half of the bootstrap bias distributions of the first convolutional layer of VGG-16 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



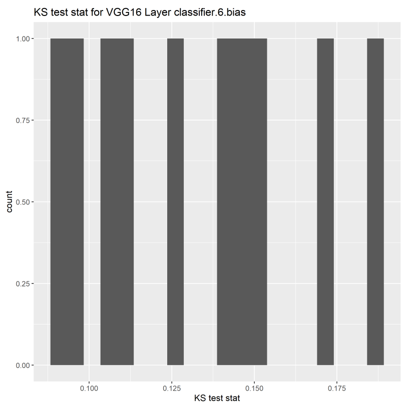
(a) KS test statistic



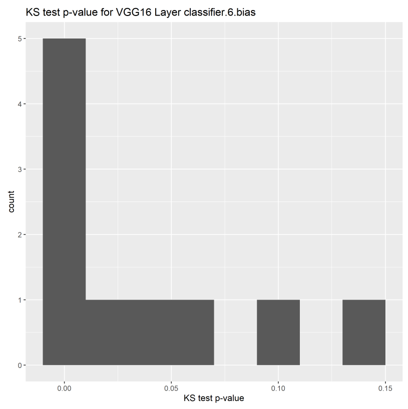
(b) KS test p-value

Figure 3.32. KS test results for selected weights in bootstrap VGG-16, final fully connected layer.

Figure 3.32 shows that over 99% of the bootstrap weight distributions of the final fully connected layer of VGG-16 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.



(a) KS test statistic



(b) KS test p-value

Figure 3.33. KS test results for selected biases in bootstrap VGG-16, final fully connected layer.

Figure 3.33 shows that over half of the bootstrap bias distributions of the final fully connected layer of VGG-16 trained on CIFAR10 are non-normal as determined by the KS test, with a p-value threshold of 0.05.

The Tables 3.3 and 3.4 summarize the KS test statistics and the p-values for the weights and the biases of the different CNN models. The Tables 3.5 and 3.6 summarize the KS test statistics and the p-values for the biases of the different CNN models.

Table 3.3. Kolmogorov-Smirnov test statistic quantiles of weights from bootstrapped neural networks. Up to 1000 weights were selected randomly from each layer.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.051	0.060	0.065	0.074	0.083	0.093	0.102	0.109	0.117
CNN3, last fc	0.035	0.041	0.046	0.055	0.070	0.088	0.106	0.119	0.138
VGG16, 1st conv	0.038	0.049	0.056	0.071	0.091	0.118	0.140	0.160	0.268
VGG16, last fc	0.124	0.146	0.157	0.177	0.207	0.237	0.264	0.275	0.301
Incept v3, 1st conv	0.057	0.075	0.096	0.164	0.249	0.366	0.427	0.445	0.469
Incept v3, 2nd conv	0.090	0.127	0.157	0.205	0.280	0.359	0.414	0.441	0.470
Incept v3, 3rd conv	0.075	0.110	0.132	0.186	0.280	0.339	0.367	0.386	0.418
Incept v3, 4th conv	0.085	0.104	0.114	0.139	0.173	0.202	0.228	0.247	0.302
Incept v3, 5th conv	0.066	0.084	0.097	0.119	0.149	0.180	0.208	0.225	0.256
Incept v3, 6th conv	0.043	0.054	0.063	0.082	0.118	0.148	0.175	0.190	0.211

Table 3.4. Kolmogorov-Smirnov test p-value quantiles of weights from bootstrapped neural networks. Selected weights match those in Table 3.3.

Layer	1st	5th	10th	25th	50th	75th	90th	95th	99th
CNN3, 1st conv	0.030	0.054	0.084	0.143	0.243	0.374	0.535	0.631	0.813
CNN3, last fc	0.006	0.027	0.065	0.185	0.439	0.728	0.901	0.955	0.991
VGG16, 1st conv	1.4e-10	5.0e-4	3.3e-3	0.022	0.134	0.390	0.686	0.819	0.971
VGG16, last fc	2.9e-13	3.8e-11	2.9e-10	2.3e-8	1.9e-6	7.6e-5	7.2e-4	1.9e-3	0.014
Incept, 1st conv	2.3e-31	3.5e-28	7.5e-26	6.9e-19	7.5e-9	4.5e-4	0.110	0.346	0.691
Incept, 2nd conv	1.8e-31	1.4e-27	2.6e-24	3.8e-18	4.1e-11	4.3e-6	9.7e-4	0.014	0.153
Incept, 3rd conv	1.1e-24	4.9e-21	5.3e-19	3.1e-16	4.1e-11	4.0e-5	9.0e-3	0.045	0.345
Incept, 4th conv	7.5e-13	1.1e-8	1.8e-7	5.8e-6	1.8e-4	4.9e-3	0.034	0.066	0.208
Incept, 5th conv	2.4e-9	2.8e-7	2.9e-6	8.0e-5	2.0e-3	0.025	0.105	0.218	0.495
Incept, 6th conv	1.8e-6	2.6e-5	1.5e-4	2.1e-3	0.026	0.244	0.563	0.749	0.928

We then examine the shape of the non-normal weights and biases from each layer of VGG16 and Inception V3. Below we create QQ plots for selected non-normal weights and biases. Note Inception V3 first six convolutional layers do not have bias parameters. Unfortunately we do not see one distribution that can fit all the non-normal weights or biases.

Figure 3.34 shows some bootstrap weight distributions are heavier tailed than normal as in (b), (c), (d) for the first convolutional layer of Incept v3. The other three bootstrap distributions become non-normal due to a handful of very large or small values.

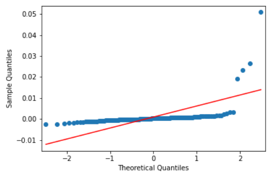
Figure 3.35 shows some bootstrap weight distributions are heavier tailed than normal as in (a), (c), (d) for the second convolutional layer of Incept v3. The other three bootstrap distributions become non-normal due to a handful of very large or small values.

Table 3.5. Kolmogorov-Smirnov test statistic quantiles of biases from bootstrapped neural networks. Up to 1000 biases were selected randomly from each layer.

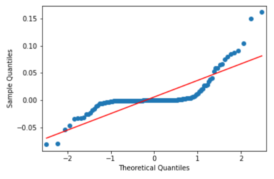
Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.048	0.051	0.053	0.0590	0.067	0.073	0.080	0.088	0.091
CNN3, last fc	0.049	0.051	0.055	0.065	0.073	0.078	0.080	0.081	0.082
VGG16, 1st conv	0.052	0.062	0.071	0.088	0.114	0.146	0.173	0.177	0.200
VGG16, last fc	0.091	0.093	0.096	0.105	0.132	0.150	0.172	0.179	0.185

Table 3.6. Kolmogorov-Smirnov test p-value quantiles of biases from bootstrapped neural networks. Selected biases match those in Table 3.5.

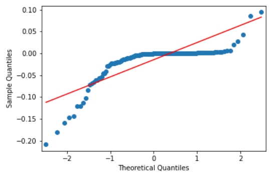
Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.156	0.184	0.271	0.379	0.497	0.652	0.779	0.802	0.864
CNN3, last fc	0.253	0.261	0.271	0.312	0.385	0.530	0.732	0.799	0.852
VGG16, 1st conv	2.6e-5	8.0e-5	1.3e-4	2.1e-3	0.029	0.163	0.377	0.560	0.755
VGG16, last fc	3.9e-5	8.7e-5	1.5e-4	1.4e-3	8.3e-3	0.055	0.100	0.121	0.137



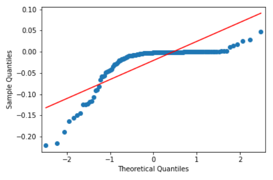
(a)



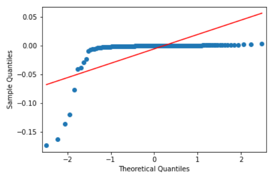
(b)



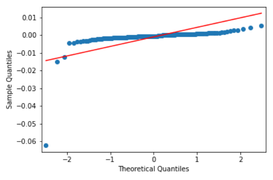
(c)



(d)



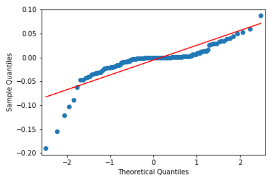
(e)



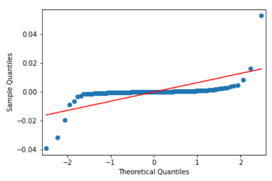
(f)

Figure 3.34. QQ Plots of individual non-normal weights selected from Incept-v3, first convolutional layer.

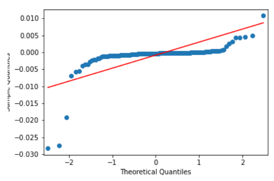
Figures 3.36, 3.37, 3.38, 3.39 shows some bootstrap weight distributions which are heavier tailed than normal for the third, fourth, fifth and sixth convolutional layers of Inception v3 trained on CIFAR10.



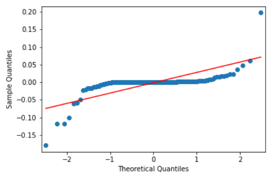
(a)



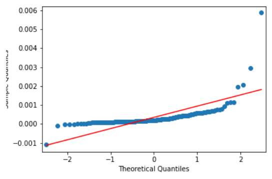
(b)



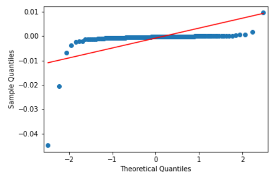
(c)



(d)

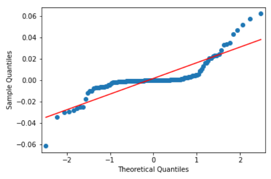


(e)

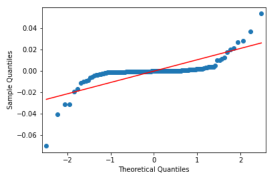


(f)

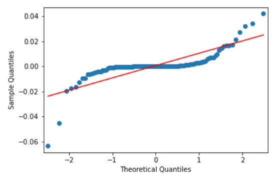
Figure 3.35. QQ Plots of individual non-normal weights selected from Incept-v3, second convolutional layer.



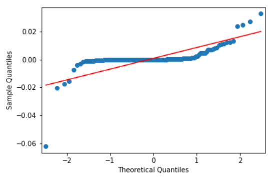
(a)



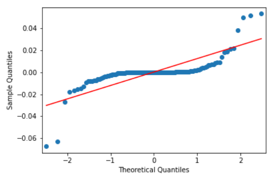
(b)



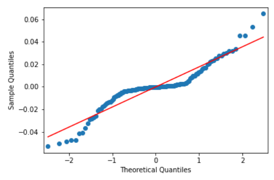
(c)



(d)



(e)



(f)

Figure 3.36. QQ Plots of individual non-normal weights selected from Incept-v3, third convolutional layer.

Figures 3.40 shows some bootstrap bias distributions which are heavier tailed than normal for the first convolutional layer of VGG-16.

Figures 3.41, 3.42 show some bootstrap weight and bias distributions which are heavier tailed than normal for the final fully connected layer of VGG16.

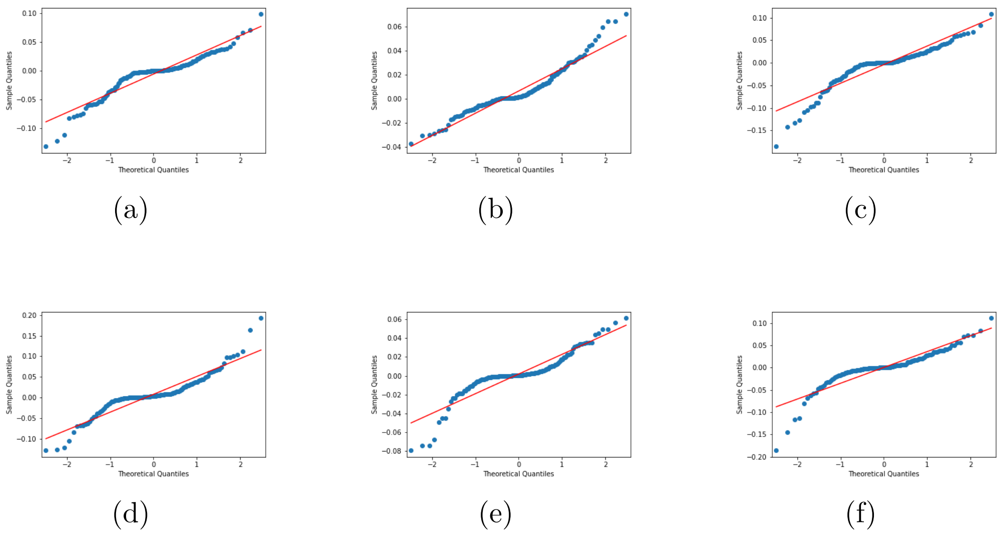


Figure 3.37. QQ Plots of individual non-normal weights selected from Incept-v3, fourth convolutional layer.

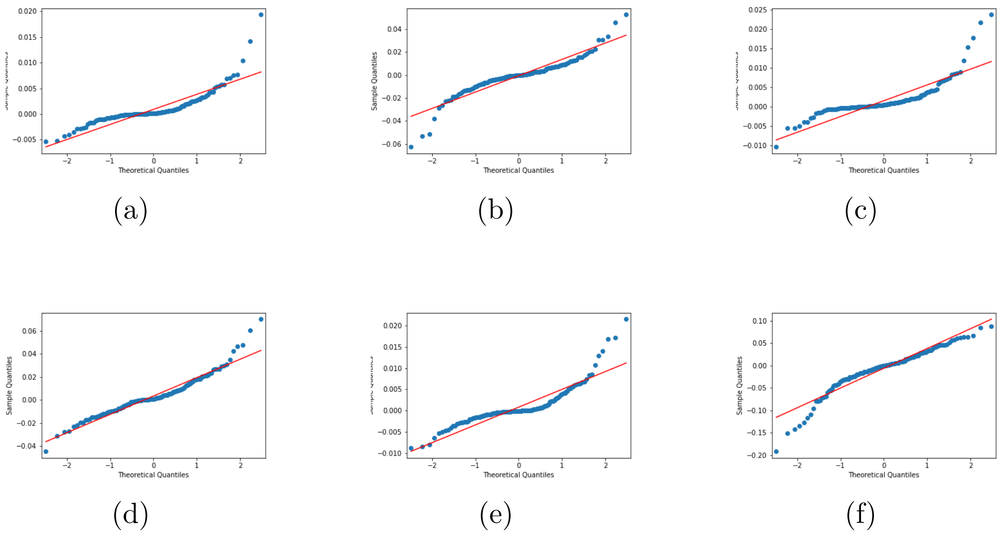


Figure 3.38. QQ Plots of individual non-normal weights selected from Incept-v3, fifth convolutional layer.

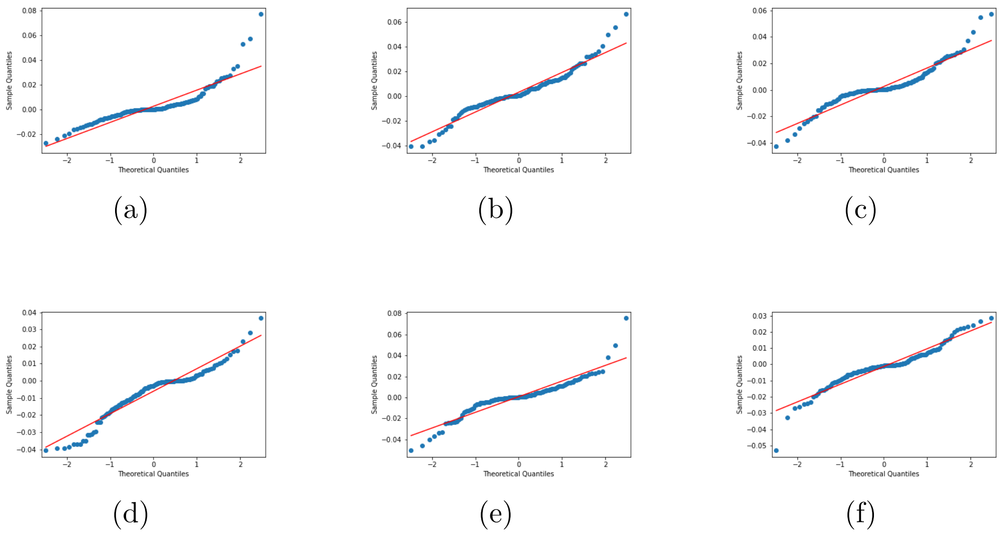


Figure 3.39. QQ Plots of individual non-normal weights selected from Incept-v3, sixth convolutional layer.

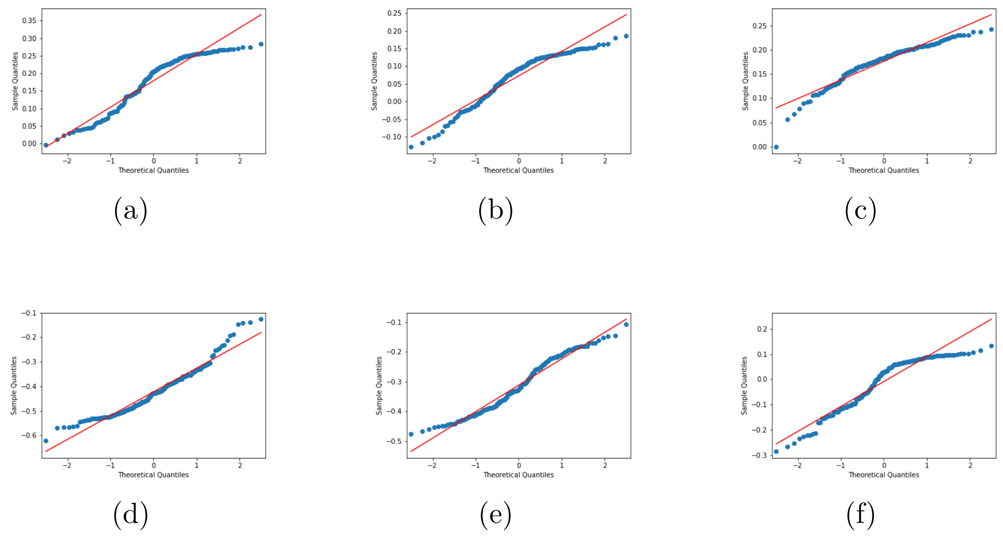
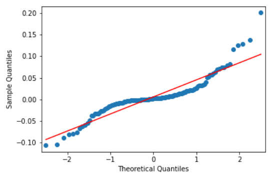
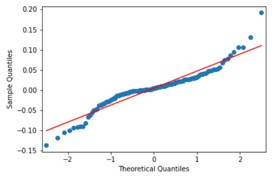


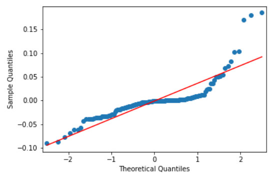
Figure 3.40. QQ Plots of individual non-normal biases selected from bootstrapped VGG-16, first convolutional layer.



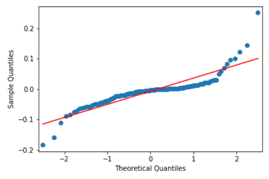
(a)



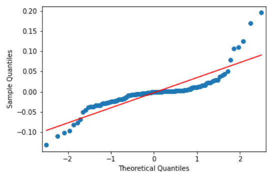
(b)



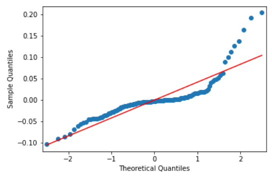
(c)



(d)

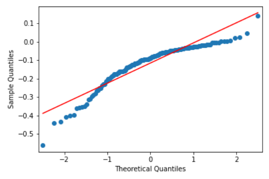


(e)

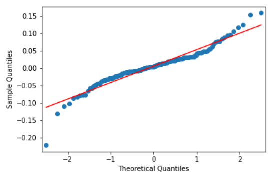


(f)

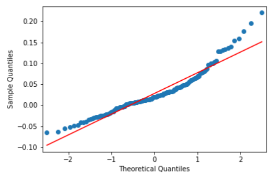
Figure 3.41. QQ Plots of individual non-normal weights selected from bootstrapped VGG-16, final fully connected layer.



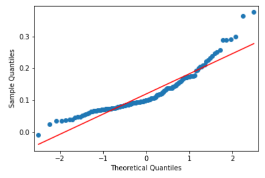
(a)



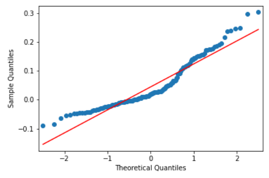
(b)



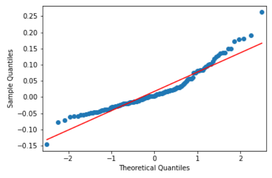
(c)



(d)



(e)



(f)

Figure 3.42. QQ Plots of individual non-normal biases selected from bootstrapped VGG-16, final fully connected layer.

3.6 CNN Model Parameter Distribution by Varying the Random Initial Seeds

In this section, we train CNN3 on MNIST, VGG16 and ResNet18 on CIFAR10 with different random initial seeds without bootstrapping the training dataset. All the models are trained 150 times. We then examine the distributions of the weights and the biases. We also compare the parameter distributions with the bootstrap distribution in the next Section.

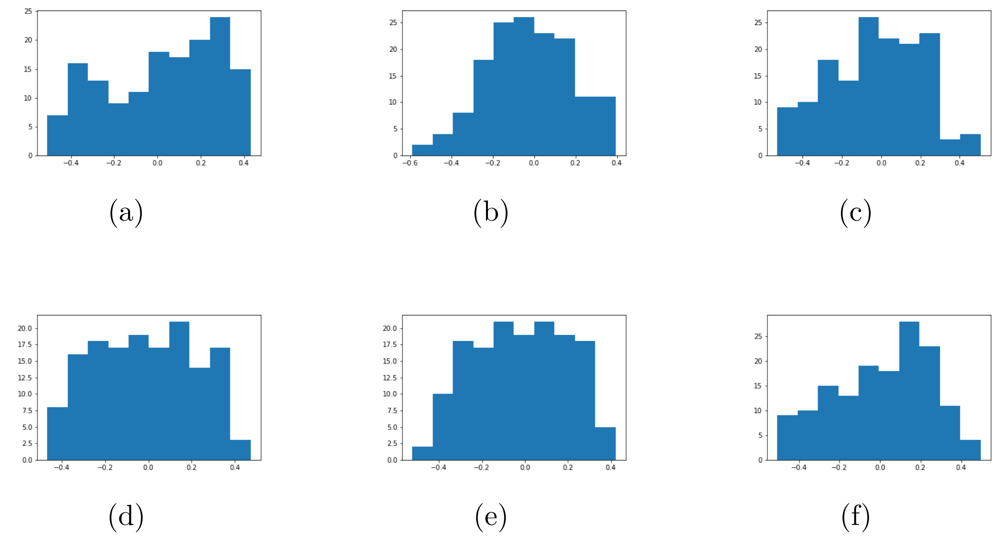
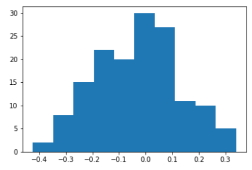
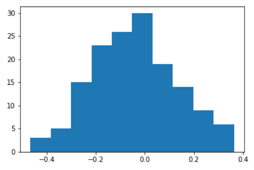


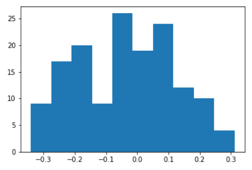
Figure 3.43. Distribution of weights selected from CNN3 generated from varying initial seed, first convolutional layer.

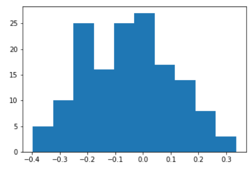
The weight distributions when varying the initial random seed of the first convolutional layer for the CNN3 model trained on MNIST data are approximately symmetric, as shown in Figure 3.43 with selected ones. The bias distributions of the fully connected layer are approximately symmetric too, as shown in Figure 3.44. Weights and biases are primarily in the +/- 0.4 range.

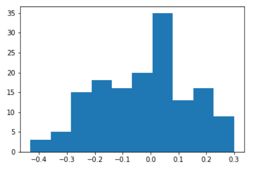
The weight distributions when varying the initial random seed of the fully connected layer for the CNN3 model trained on MNIST data are approximately symmetric, as shown in Figure 3.45 with selected ones. The bias distributions of the fully connected layer are approximately symmetric too, as shown in Figure 3.46. Weights are primarily in the +/- 0.15 range, while biases are in the +/- 0.05 range.


(a)


(b)


(c)


(d)


(e)

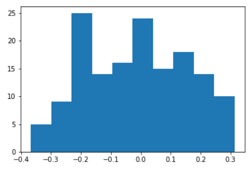
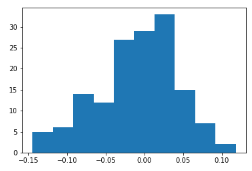
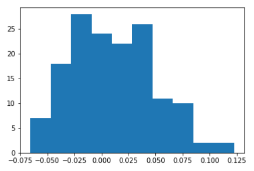
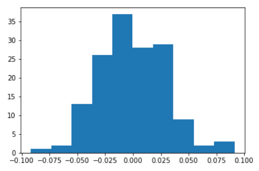
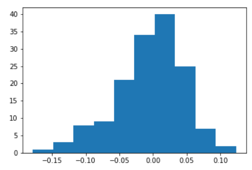

(f)

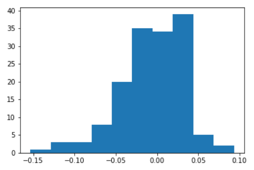
Figure 3.44. Distribution of biases selected from CNN3 generated from varying initial seed, first convolutional layer.


(a)


(b)


(c)


(d)


(e)

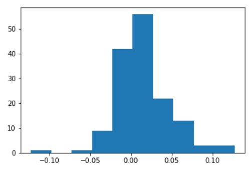

(f)

Figure 3.45. Distribution of weights selected from CNN3 generated from varying initial seed, fully connected layer.

The weight distributions when varying the initial random seed of the first convolutional layer for the VGG16 model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.47 with selected ones. Weights are primarily in the +/- 0.1 range. The bias distributions of the fully connected layer are approximately symmetric too, as shown in

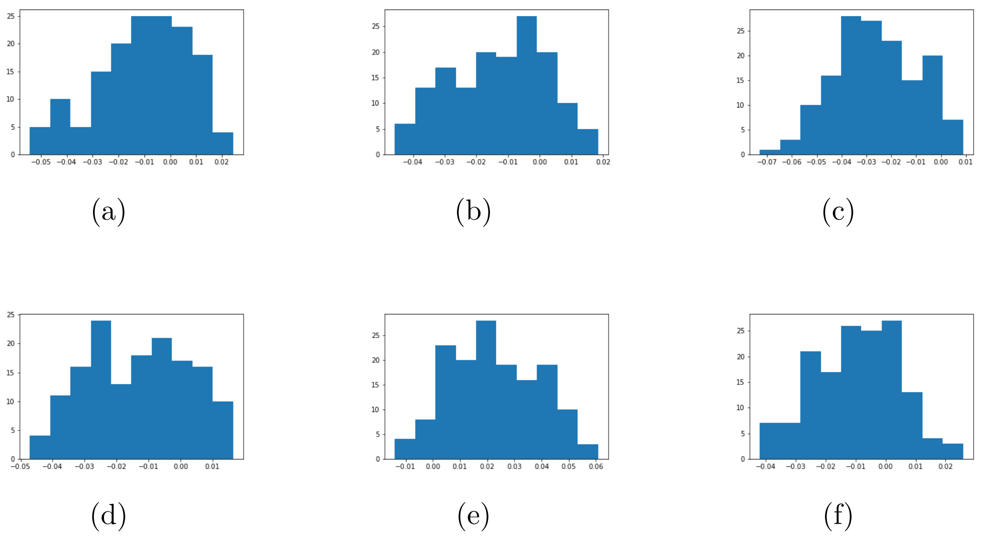


Figure 3.46. Distribution of biases selected from CNN3 generated from varying initial seed, fully connected layer.

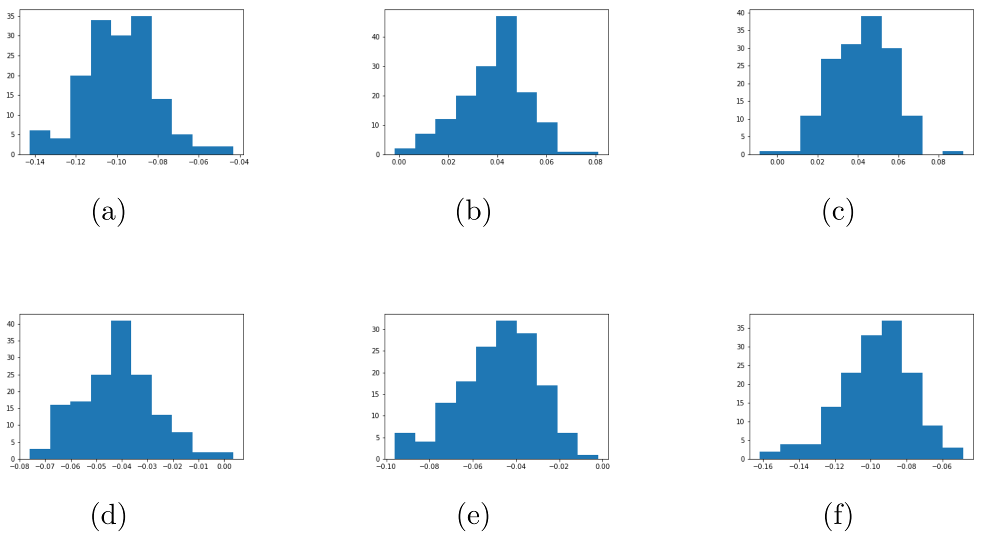
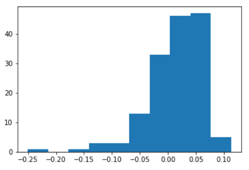
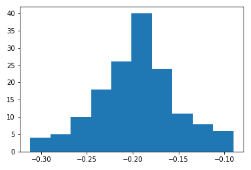


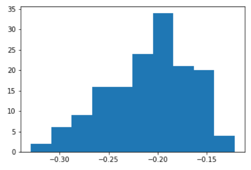
Figure 3.47. Distribution of weights selected from VGG-16 generated from varying initial seed, first convolutional layer.

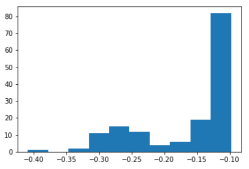
Figure 3.48, with the exception of sub-figure (d). Many of the biases shown are centered on -0.2, and in the -0.3/-0.1 range.

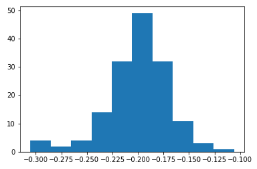
The weight distributions when varying the initial random seed of the final fully connected layer for the VGG16 model trained on CIFAR10 data are approximately symmetric, with the exception of a few extreme values, as shown in Figure 3.49 with selected ones. The


(a)


(b)


(c)


(d)


(e)

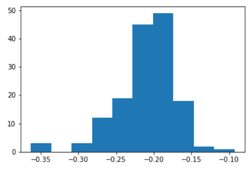
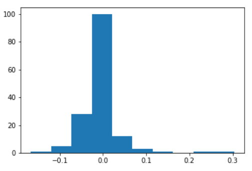
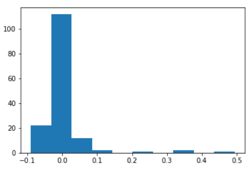
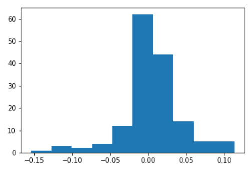
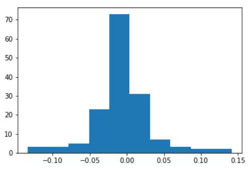

(f)

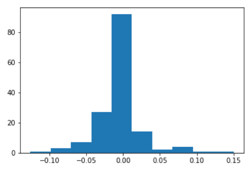
Figure 3.48. Distribution of biases selected from VGG-16 generated from varying initial seed, first convolutional layer.


(a)


(b)


(c)


(d)


(e)

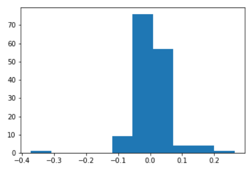

(f)

Figure 3.49. Distribution of weights selected from VGG-16 generated from varying initial seed, final fully connected layer.

bias distributions of the fully connected layer are approximately symmetric too, as shown in Figure 3.50. Weights and biases are primarily in the +/- 0.3 range.

The weight distributions when varying the initial random seed of the first convolutional layer for the ResNet model trained on CIFAR10 data are approximately symmetric, with

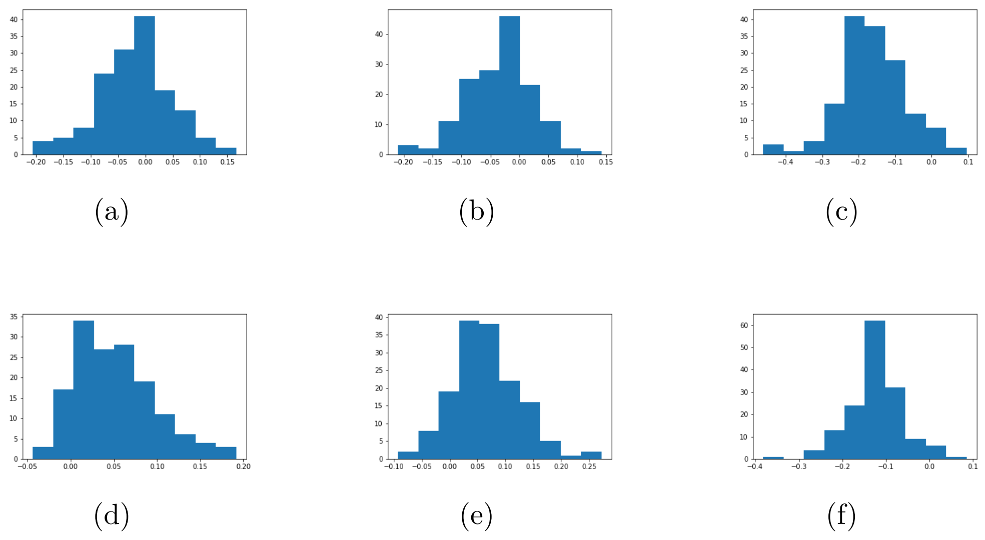


Figure 3.50. Distribution of biases selected from VGG-16 generated from varying initial seed, final fully connected layer.

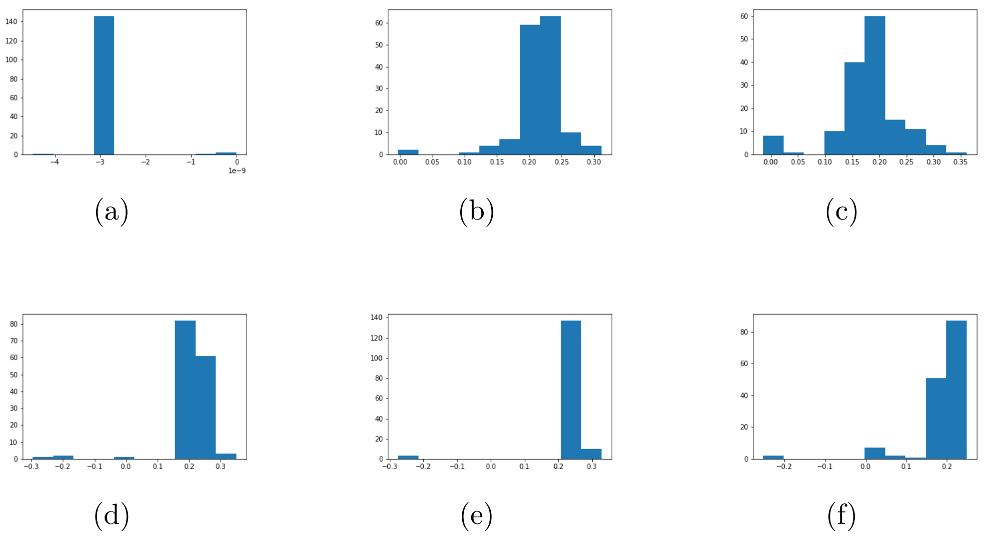


Figure 3.51. Distribution of weights selected from ResNet-18 generated from varying initial seed, first convolutional layer.

the exception of a few extreme values, as shown in Figure 3.51 with selected ones. Weights are primarily in the +/- 0.3 range.

The weight distributions when varying the initial random seed of the final fully connected layer for the ResNet model trained on CIFAR10 data are approximately symmetric, as shown in Figure 3.52 with selected ones. The bias distributions of the fully connected layer are

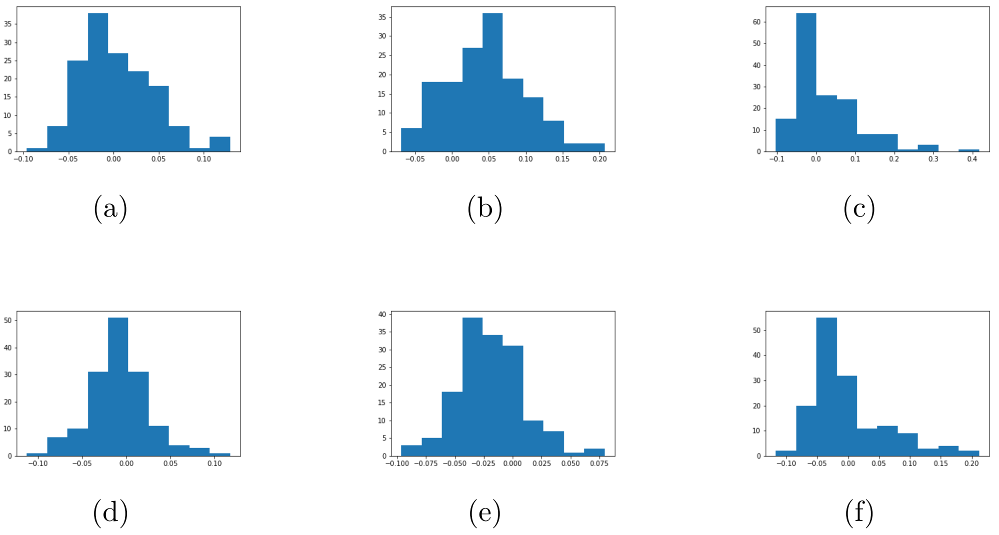


Figure 3.52. Distribution of weights selected from ResNet-18 generated from varying initial seed, fully connected layer.

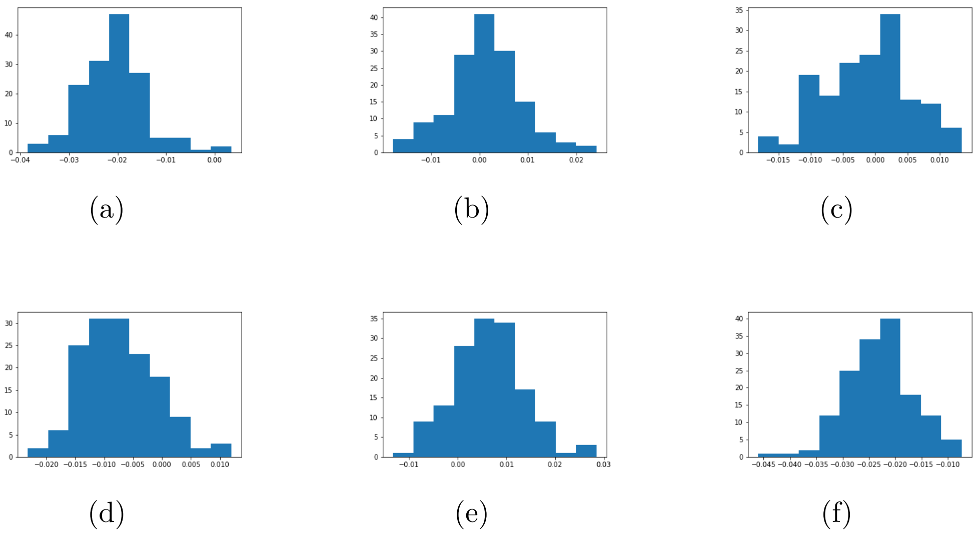


Figure 3.53. Distribution of biases selected from ResNet-18 generated from varying initial seed, fully connected layer.

approximately symmetric too, as shown in Figure 3.53. Weights are primarily in the +/- 0.2 range. Biases are primarily in the +/- 0.03 range.

Table 3.7. Kolmogorov-Smirnov test statistic quantiles from neural networks generated by varying the initial random seed. Up to 1000 weights were selected from each layer. For layers which are also represented in Table 3.3, the same weights are represented in both tables. For layers not represented in Table 3.3, the weights were selected randomly.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.045	0.059	0.064	0.073	0.084	0.092	0.101	0.107	0.119
CNN3, last fc	0.033	0.040	0.045	0.054	0.069	0.086	0.106	0.117	0.137
VGG16, 1st conv	0.034	0.040	0.045	0.056	0.072	0.094	0.131	0.176	0.266
VGG16, last fc	0.121	0.141	0.151	0.174	0.203	0.232	0.256	0.275	0.297
ResNet18, 1st conv	0.035	0.045	0.052	0.066	0.092	0.149	0.477	0.477	0.477
ResNet18, last fc	0.038	0.046	0.054	0.069	0.089	0.117	0.142	0.156	0.185

Table 3.8. Kolmogorov-Smirnov test p-value quantiles from neural networks generated by varying the initial random seeds. Selected weights match those in Table 3.7.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.026	0.058	0.088	0.148	0.230	0.385	0.559	0.655	0.911
CNN3, last fc	0.006	0.031	0.064	0.207	0.448	0.763	0.905	0.962	0.996
VGG16, 1st conv	5.5e-10	1.4e-4	9.8e-3	0.124	0.385	0.710	0.908	0.961	0.992
VGG16, last fc	2.3e-12	1.1e-10	2.9e-9	1.2e-7	5.9e-6	1.8e-4	1.7e-3	4.1e-3	0.022
ResNet18, 1st conv	6.9e-32	7.0e-32	7.2e-32	2.2e-3	0.150	0.503	0.795	0.906	0.990
ResNet18, last fc	5.9e-5	1.1e-3	4.2e-3	0.031	0.175	0.448	0.745	0.887	0.978

Figure 3.54 shows that over 95% of the weight distributions of the first convolutional layer of CNN3 trained on MNIST, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.

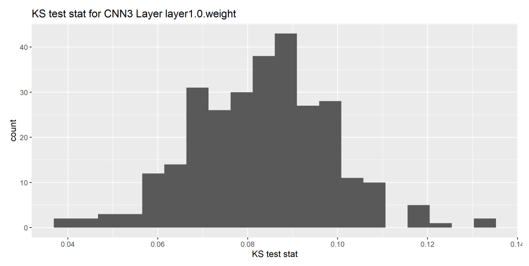
Figure 3.55 shows that over 99% of the bias distributions of the first convolutional layer of CNN3 trained on MNIST, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.

Table 3.9. Kolmogorov-Smirnov test statistic quantiles of bias values from neural networks generated by varying the initial random seeds. Up to 1000 biases were selected from each layer. For layers which are also represented in Table 3.5, the same biases are represented in both tables. For layers not represented in Table 3.5, the biases were selected randomly.

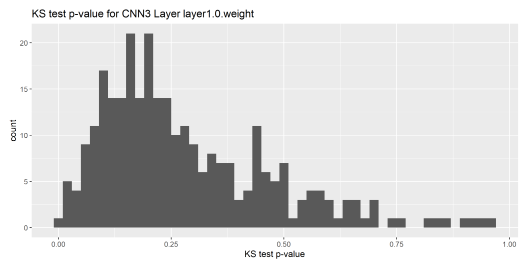
Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.043	0.044	0.048	0.057	0.068	0.078	0.084	0.090	0.097
CNN3, last fc	0.048	0.050	0.052	0.056	0.060	0.069	0.080	0.087	0.093
VGG16, 1st conv	0.036	0.042	0.058	0.069	0.099	0.143	0.180	0.212	0.256
VGG16, last fc	0.045	0.047	0.048	0.053	0.067	0.072	0.083	0.093	0.102
ResNet18, last fc	0.039	0.039	0.040	0.047	0.055	0.061	0.071	0.072	0.072

Table 3.10. Kolmogorov-Smirnov test p-value quantiles of bias values from neural networks generated by varying the initial random seeds. Selected biases match those in Table 3.9.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.112	0.172	0.228	0.312	0.476	0.698	0.867	0.920	0.930
CNN3, last fc	0.145	0.203	0.276	0.461	0.628	0.706	0.788	0.826	0.857
VGG16, 1st conv	8.5e-9	1.9e-6	9.7e-5	3.5e-3	0.096	0.438	0.655	0.936	0.979
VGG16, last fc	0.087	0.158	0.246	0.392	0.492	0.770	0.854	0.879	0.900
ResNet18, last fc	0.392	0.401	0.413	0.619	0.736	0.875	0.965	0.969	0.972



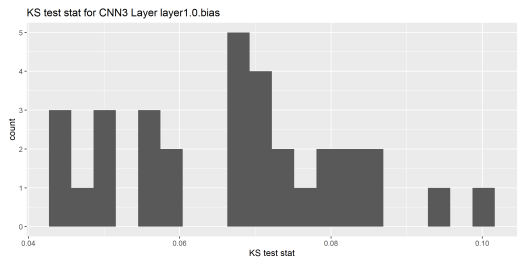
(a) KS test statistic



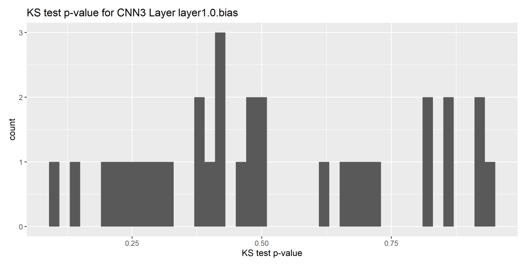
(b) KS test p-value

Figure 3.54. KS test results for selected weights when varying the initial random seed in CNN3, first convolutional layer.

Figure 3.56 shows that over 90% of the weight distributions of the fully connected layer of CNN3 trained on MNIST, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.

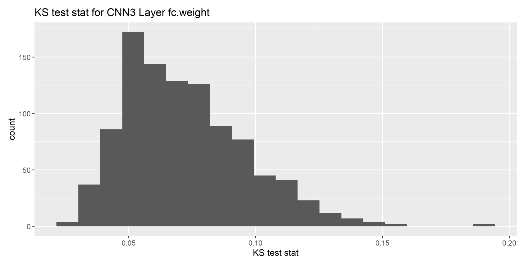


(a) KS test statistic

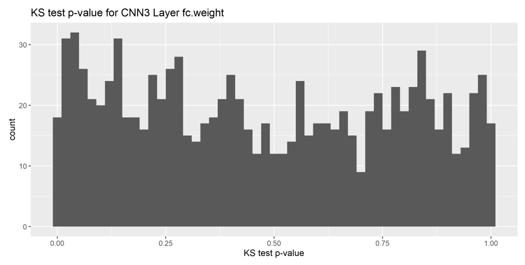


(b) KS test p-value

Figure 3.55. KS test results for selected biases when varying the initial random seed in CNN3, first convolutional layer.

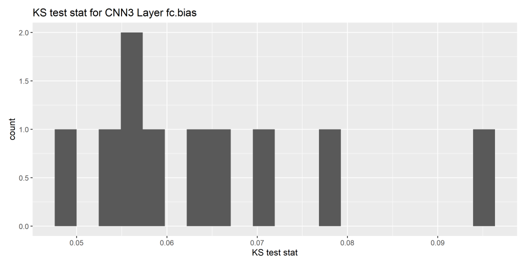


(a) KS test statistic

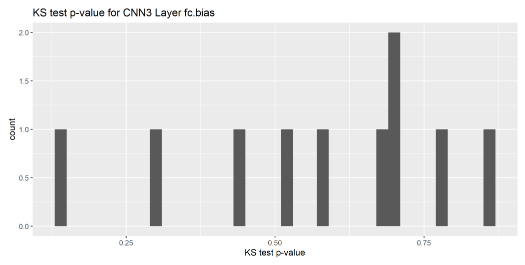


(b) KS test p-value

Figure 3.56. KS test results for selected weights when varying the initial random seed in CNN3, fully connected layer.



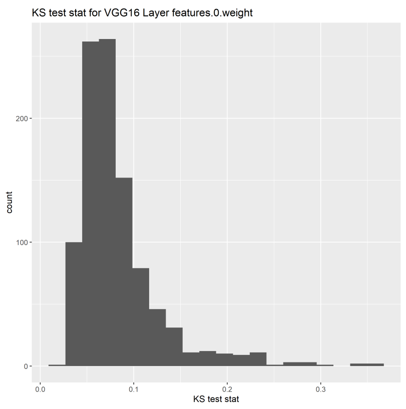
(a) KS test statistic



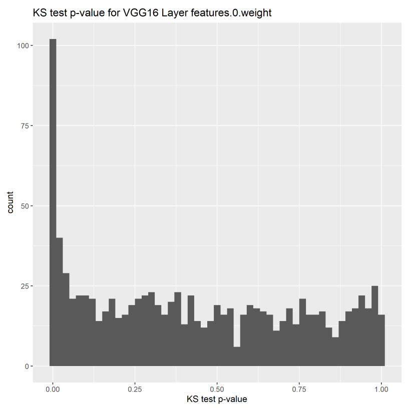
(b) KS test p-value

Figure 3.57. KS test results for selected biases when varying the initial random seed in CNN3, fully connected layer.

Figure 3.57 shows that all 10 of the bias distributions of the fully connected layer of CNN3 trained on MNIST, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.



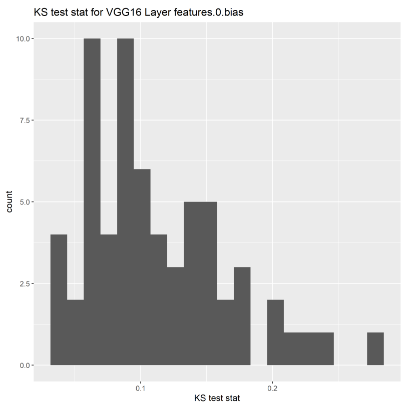
(a) KS test statistic



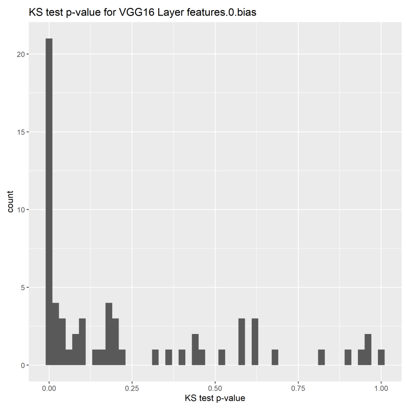
(b) KS test p-value

Figure 3.58. KS test results for selected weights when varying the initial random seed in VGG16, first convolutional layer.

Figure 3.58 shows that over 75% of the weight distributions of the first convolutional layer of VGG16 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.



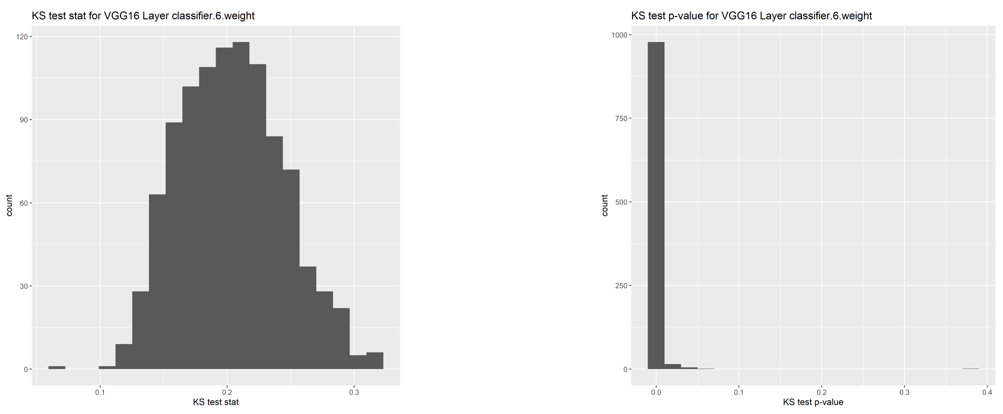
(a) KS test statistic



(b) KS test p-value

Figure 3.59. KS test results for selected biases when varying the initial random seed in VGG-16, first convolutional layer.

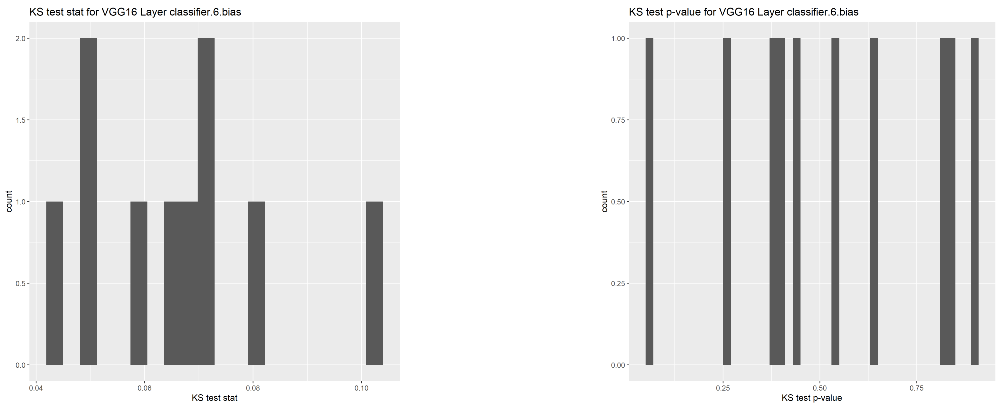
Figure 3.59 shows that over half of the bias distributions of the first convolutional layer of VGG16 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.



(a) KS test statistic (b) KS test p-value

Figure 3.60. KS test results for selected weights when varying the initial random seed in VGG16, final fully connected layer.

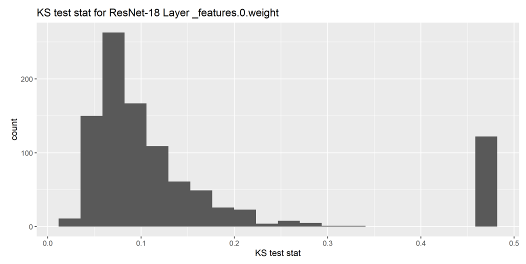
Figure 3.60 shows that over 99% of the weight distributions of the final fully connected layer of VGG16 trained on CIFAR10, by varying initial seed, are non-normal as determined by the KS test, with a p-value threshold of 0.05.



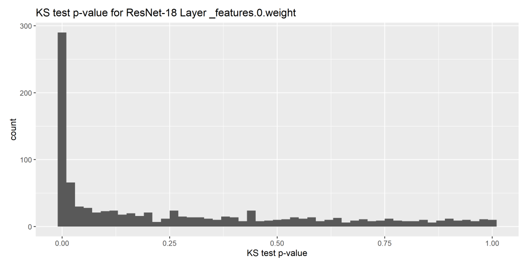
(a) KS test statistic (b) KS test p-value

Figure 3.61. KS test results for selected biases when varying the initial random seed in VGG-16, final fully connected layer.

Figure 3.61 shows that all 10 of the bias distributions of the final fully connected layer of VGG16 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.



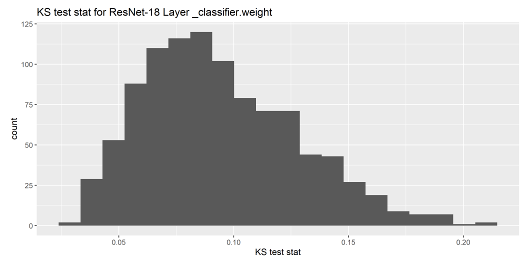
(a) KS test statistic



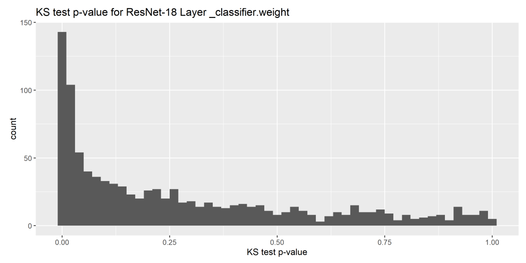
(b) KS test p-value

Figure 3.62. KS test results for selected weights when varying the initial random seed in ResNet-18, first convolutional layer.

Figure 3.62 shows that over half of the weight distributions of the first convolutional layer of ResNet-18 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.



(a) KS test statistic

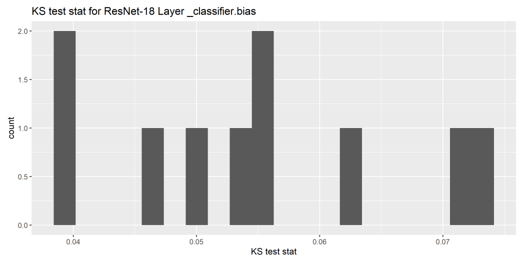


(b) KS test p-value

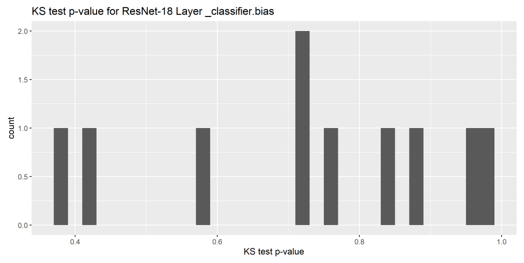
Figure 3.63. KS test results for selected weights when varying the initial random seed in ResNet-18, fully connected layer.

Figure 3.63 shows that over half of the weight distributions of the fully connected layer of ResNet-18 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.

Figure 3.64 shows that all 10 of the bias distributions of the fully connected layer of ResNet-18 trained on CIFAR10, by varying initial seed, are normal as determined by the KS test, with a p-value threshold of 0.05.

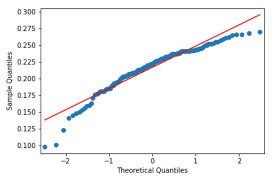


(a) KS test statistic

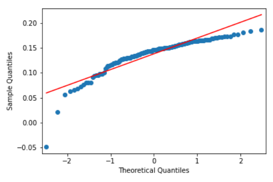


(b) KS test p-value

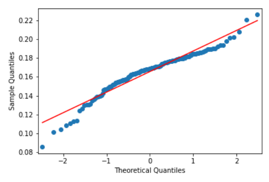
Figure 3.64. KS test results for selected biases when varying the initial random seed in ResNet-18, fully connected layer.



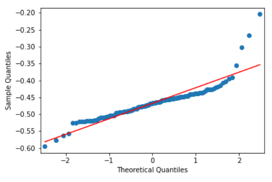
(a)



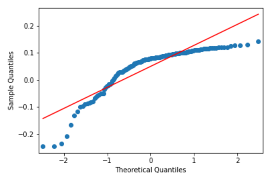
(b)



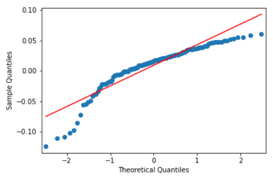
(c)



(d)

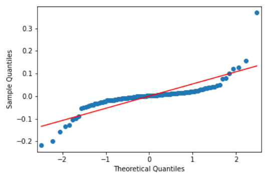


(e)

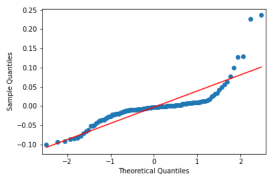


(f)

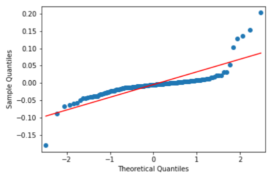
Figure 3.65. QQ Plots of individual non-normal biases selected from VGG-16 varying initial seeds, first convolutional layer.



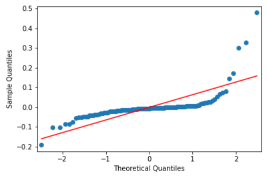
(a)



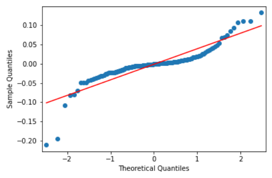
(b)



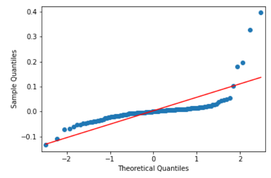
(c)



(d)

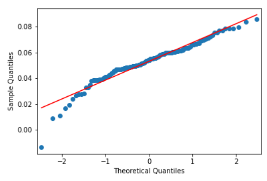


(e)

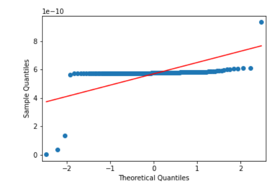


(f)

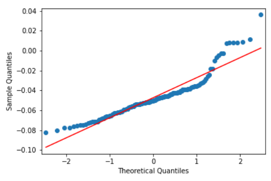
Figure 3.66. QQ Plots of individual non-normal weights selected from VGG-16 varying initial seeds, final fully connected layer.



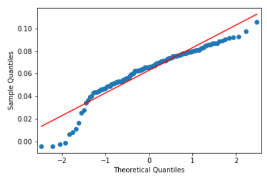
(a)



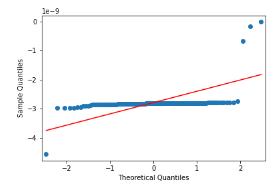
(b)



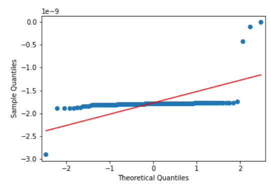
(c)



(d)

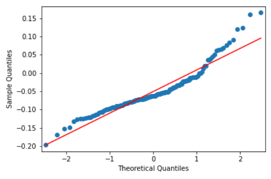


(e)

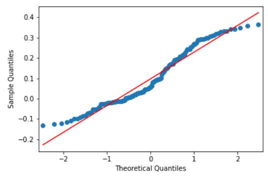


(f)

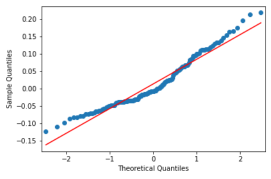
Figure 3.67. QQ Plots of individual non-normal weights selected from ResNet-18 varying the initial random seed, first convolutional layer.



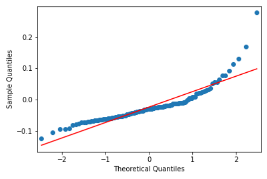
(a)



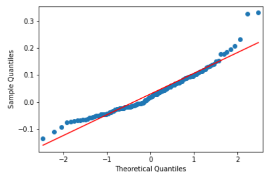
(b)



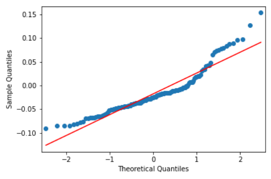
(c)



(d)



(e)



(f)

Figure 3.68. QQ Plots of individual non-normal weights selected from ResNet-18 varying the initial random seed, fully connected layer.

3.7 Distance Between Two Distributions

The weights and the biases selected in the section for varying the random initial seeds, 3.6, match the parameters selected in the bootstrap section, 3.5. Hence we can compute the distance between two sets of selected parameters from different training procedures. Distances are evaluated using the Wasserstein 1 distance, as well as the two sample Kolmogorov-Smirnov test.

The Wasserstein distance [70] is a metric based on the real world task of moving dirt. It is defined as the amount of earth that would need to be moved multiplied by the distance it would move, in order to turn one pile of earth into a different pile of earth. Here we use the one-dimensional metric to compare the distributions of weights and biases derived from bootstrapping neural networks with those derived from networks generated by varying initial seeds. The Wasserstein 1-distance, $W_1(u, v)$ for probability measures $u, v \in P_p(\mathbb{R})$ is

$$W_1(u,v) = \inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}\times\mathbb{R}} |x-y| d\boldsymbol{\pi}(x,y) \tag{3.4}$$

where $\Gamma(u, v)$ is the set of probability distributions on $\mathbb{R} \times \mathbb{R}$ whose marginals are u and v. This definition is equivalent [71] to

$$W_1(u,v) = \int_{\mathbb{R}} |U - V| \tag{3.5}$$

where U and V are the cumulative distribution functions of u and v, respectively.

Figures 3.69 and 3.70 show the distance between the weight distributions are within 0.1 and the distance between the bias distributions are within 0.035, for the first convolutional layer of CNN3.

Figures 3.71 and 3.72 show the distance between the weight distributions are within 0.025 and the distance between the bias distributions are within 0.005, for the fully connected layer of CNN3.

Figures 3.73 and 3.74 show the distance between the weight and bias distributions are within 0.1 for the first convolutional layer of VGG16.

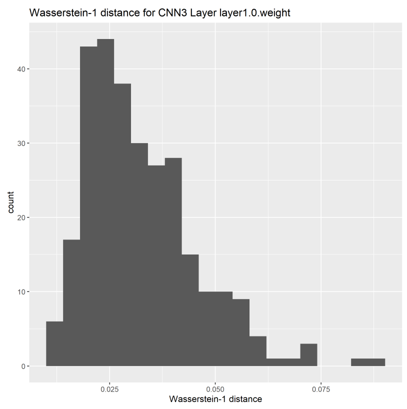


Figure 3.69. Histogram of Wasserstein 1 distance for selected weights in CNN3, first convolution layer.

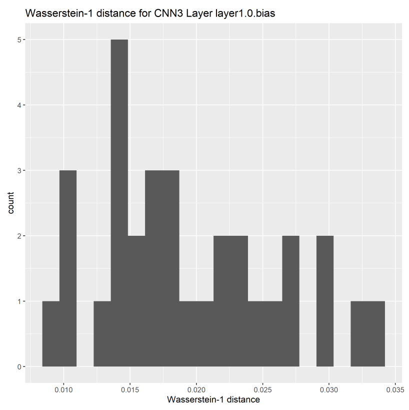


Figure 3.70. Histogram of Wasserstein 1 distance for selected biases in CNN3, first convolution layer.

Figures 3.75 and 3.76 show the distance between the weight and bias distributions are within 0.2 and the distance between the bias distributions are within 0.1, for the final fully connected layer of VGG16.

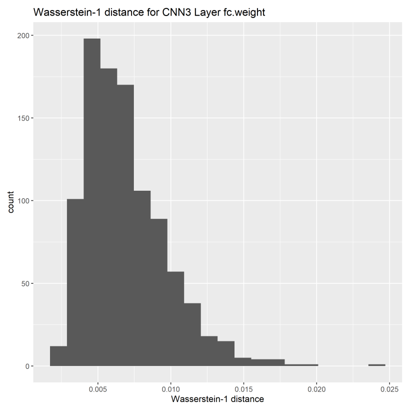


Figure 3.71. Histogram of Wasserstein 1 distance for selected weights in CNN3, fully connected (fc) layer.

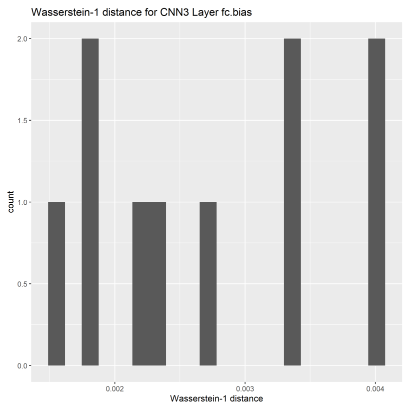


Figure 3.72. Histogram of Wasserstein 1 distance for selected biases in CNN3, fully connected layer.

Table 3.11. Wasserstein 1 distance between weights neural networks generated by varying the initial random seeds and bootstrapped neural networks. Up to 1000 weights were selected from each layer, with weights matching those represented in Tables 3.3, 3.7.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.013	0.017	0.019	0.023	0.029	0.040	0.050	0.056	0.071
CNN3, last fc	0.003	0.004	0.004	0.005	0.006	0.008	0.011	0.012	0.016
VGG16, 1st conv	0.002	0.004	0.005	0.008	0.014	0.023	0.034	0.044	0.064
VGG16, last fc	0.004	0.004	0.005	0.006	0.007	0.009	0.010	0.011	0.014

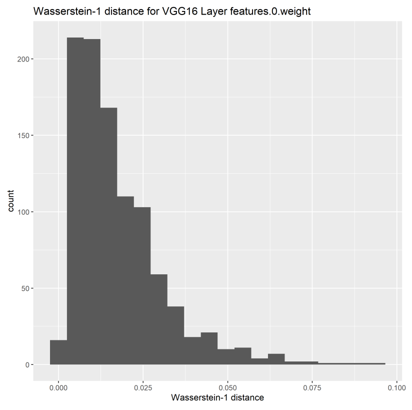


Figure 3.73. Histogram of Wasserstein 1 distance for selected weights in VGG16, first convolution layer.

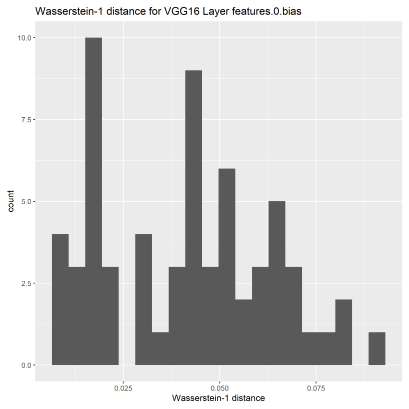


Figure 3.74. Histogram of Wasserstein 1 distance for selected biases in VGG16, first convolution layer.

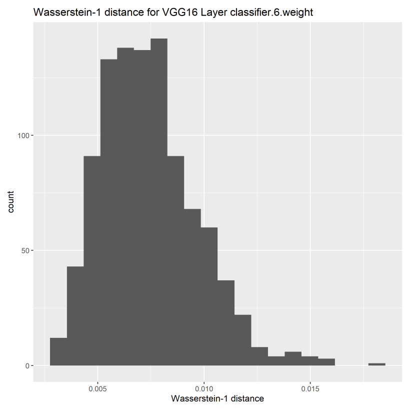


Figure 3.75. Histogram of Wasserstein 1 distance for selected weights in VGG16, final fully connected (fc) layer.

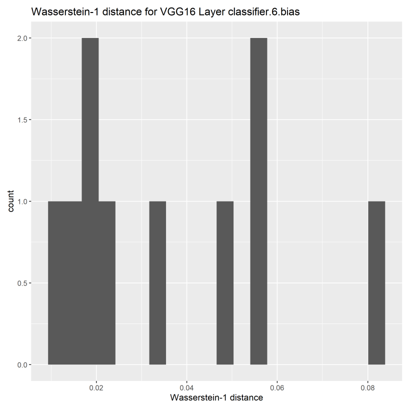


Figure 3.76. Histogram of Wasserstein 1 distance for selected biases in CNN3, first convolution layer.

Table 3.12. Wasserstein 1 distance between bias values of neural networks generated by varying the initial random seeds and bootstrapped neural networks. Up to 1000 biases were selected from each layer, with biases matching those represented in Tables 3.5, 3.9.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	9.2e-3	9.9e-3	0.010	0.014	0.018	0.024	0.030	0.031	0.033
CNN3, last fc	1.6e-3	1.7e-3	1.8e-3	1.9e-3	2.5e-3	3.4e-3	4.0e-3	4.0e-3	4.0e-3
VGG16, 1st conv	8.5e-3	0.011	0.015	0.019	0.044	0.058	0.069	0.075	0.085
VGG16, last fc	0.011	0.012	0.014	0.017	0.029	0.054	0.059	0.070	0.079

Here we also perform the KS test to measure the distance between two empirical samples. The test statistic is a quantification of the distance between the empirical distributions of two samples. The test statistic is the largest observed difference between the two empirical cumulative distributions. KS test statistics are therefore bounded between 0 and 1. Two identical distributions will observe a value of 0.

The null hypothesis of the test is that the two empirical samples are drawn from the same distribution. A cutoff of largest allowable distance is defined for a given confidence level. If the largest distance exceeds that value, the null hypothesis is rejected.

Formally, the Kolmogorov-Smirnov statistic, $D_{n,m}$, for two samples of sizes n and m, respectively, is computed as

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)| \tag{3.6}$$

where $F_{1,n}(x)$ is the empirical cumulative distribution of first sample, and $F_{1,m}(x)$ is the empirical cumulative distribution of second. The empirical cumulative distributions are defined as in Equation 3.2.

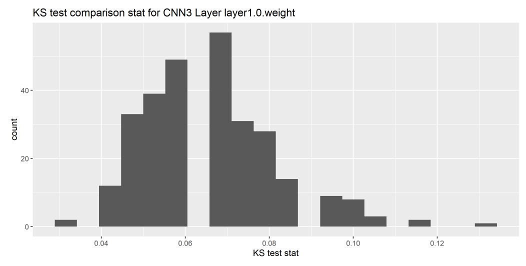
All tests in this work are evaluated at level $\alpha = 0.05$. For n and m sufficiently large, the null hypothesis that the two samples come from the same distribution is rejected if

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{nm}} \tag{3.7}$$

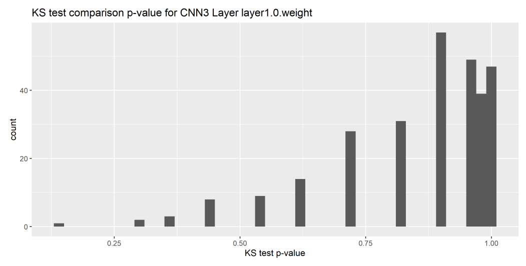
where $c(\alpha)$ is the inverse of the Kolmogorov distribution, K and is estimated formulaically by

$$c(\alpha) = \sqrt{-ln(\frac{\alpha}{2}) * \frac{1}{2}} \tag{3.8}$$

Figure 3.77 shows that over 99% of the weight distributions of the first convolutional layer of CNN3 trained on MNIST, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.

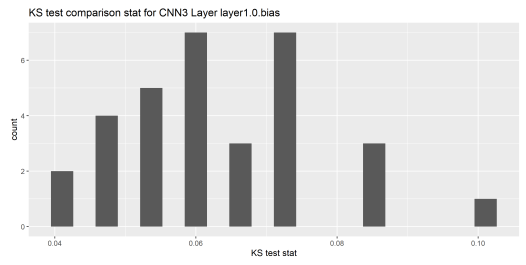


(a) KS test statistic

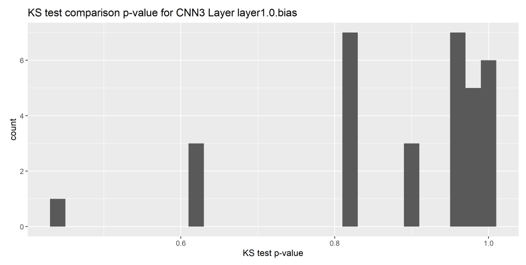


(b) KS test p-value

Figure 3.77. 2 sample KS test results for selected weights in CNN3, first convolutional layer.



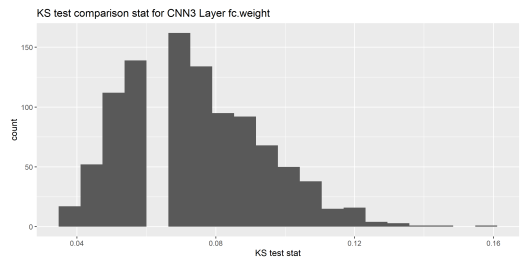
(a) KS test statistic



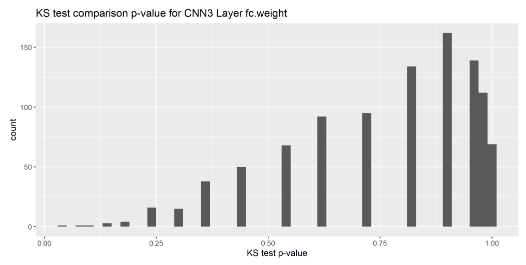
(b) KS test p-value

Figure 3.78. 2 sample KS test results for selected biases in CNN3, first convolutional layer.

Figure 3.78 shows that over 99% of the bias distributions of the first convolutional layer of CNN3 trained on MNIST, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



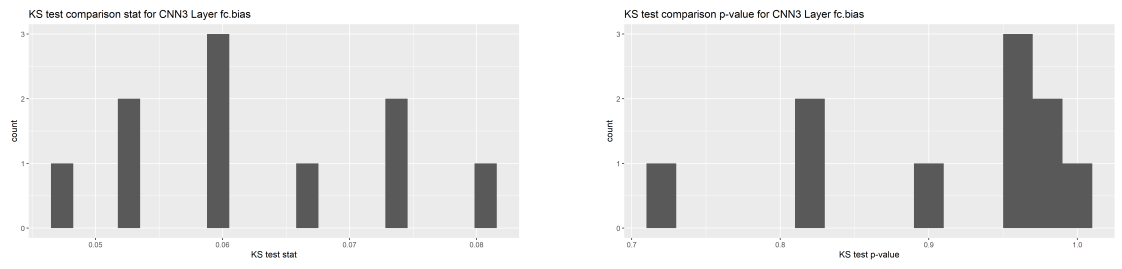
(a) KS test statistic



(b) KS test p-value

Figure 3.79. 2 sample KS test results for selected weights in CNN3, fully connected layer.

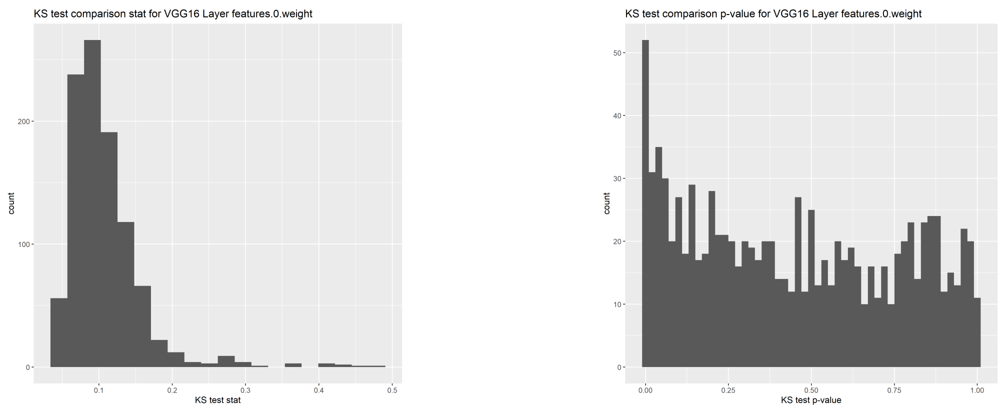
Figure 3.79 shows that over 99% of the weight distributions of the fully connected layer of CNN3 trained on MNIST, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



(a) KS test statistic (b) KS test p-value

Figure 3.80. 2 sample KS test results for selected biases in CNN3, fully connected layer.

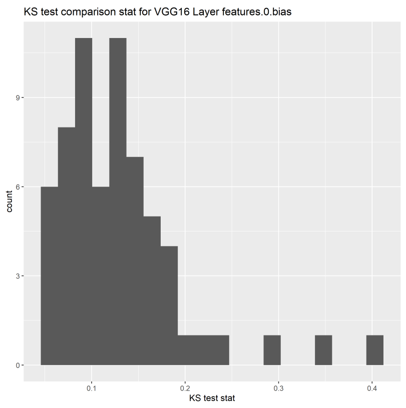
Figure 3.80 shows that all 10 of the bias distributions of the fully connected layer of CNN3 trained on MNIST, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



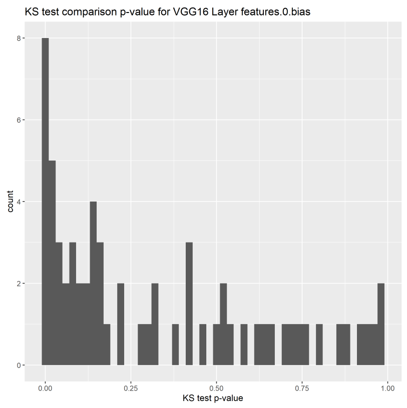
(a) KS test statistic (b) KS test p-value

Figure 3.81. 2 sample KS test results for selected weights in VGG-16, first convolutional layer.

Figure 3.81 shows that over 75% of the weight distributions of the first convolutional layer of VGG16 trained on CIFAR10, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



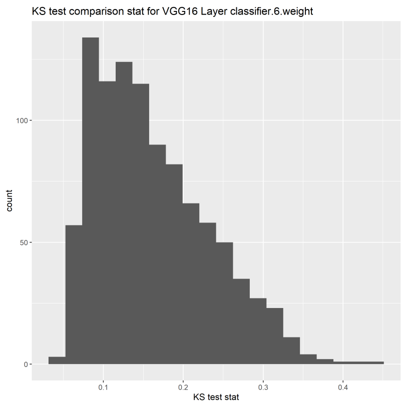
(a) KS test statistic



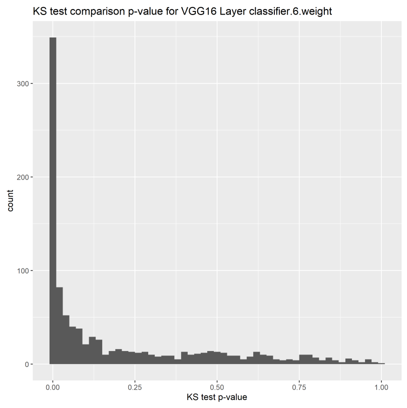
(b) KS test p-value

Figure 3.82. 2 sample KS test results for selected biases in VGG-16, first convolutional layer.

Figure 3.82 shows that over 75% of the bias distributions of the first convolutional layer of VGG16 trained on CIFAR10, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



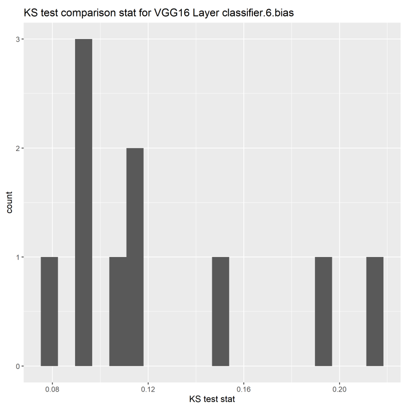
(a) KS test statistic



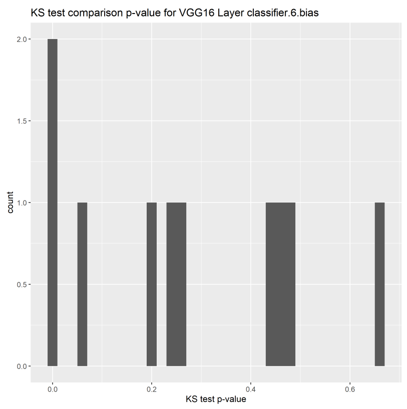
(b) KS test p-value

Figure 3.83. 2 sample KS test results for selected weights in VGG-16, final fully connected layer.

Figure 3.83 shows that over half of the weight distributions of the final fully connected layer of VGG16 trained on CIFAR10, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.



(a) KS test statistic



(b) KS test p-value

Figure 3.84. 2 sample KS test results for selected biases in VGG-16, final fully connected layer.

Figure 3.83 shows that over 75% of the weight distributions of the final fully connected layer of VGG16 trained on CIFAR10, fail to reject the null hypothesis of the 2-sample KS test, with a p-value threshold of 0.05.

Table 3.13. Kolmogorov-Smirnov 2-sample test statistic quantiles between weights generated by varying the initial random seeds and bootstrapped neural networks. Up to 1000 weights were selected from each layer, with weights matching those represented in Tables 3.3, 3.7.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.040	0.047	0.047	0.053	0.067	0.073	0.087	0.093	0.108
CNN3, last fc	0.040	0.047	0.053	0.060	0.073	0.087	0.100	0.107	0.120
VGG16, 1st conv	0.047	0.056	0.064	0.075	0.097	0.125	0.157	0.182	0.312
VGG16, last fc	0.059	0.072	0.082	0.104	0.147	0.209	0.267	0.298	0.338

Table 3.14. Kolmogorov-Smirnov 2-sample test p-value quantiles between weights generated by varying the initial random seeds and bootstrapped neural networks. Selected weights match those in Table 3.13.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.352	0.532	0.628	0.816	0.894	0.984	0.997	0.997	0.999
CNN3, last fc	0.230	0.361	0.443	0.628	0.816	0.951	0.984	0.997	0.999
VGG16, 1st conv	3.4e-7	9.5e-3	0.038	0.158	0.419	0.727	0.882	0.951	0.991
VGG16, last fc	2.2e-8	1.3e-6	2.1e-5	1.8e-3	0.061	0.334	0.631	0.779	0.927

Table 3.15. Kolmogorov-Smirnov 2-sample test statistic quantiles between biases generated by varying the initial random seeds and bootstrapped neural networks. Up to 1000 biases were selected from each layer, with biases matching those represented in Tables 3.5, 3.9.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.040	0.044	0.047	0.053	0.060	0.073	0.085	0.087	0.096
CNN3, last fc	0.047	0.050	0.053	0.055	0.060	0.072	0.074	0.077	0.079
VGG16, 1st conv	0.050	0.058	0.067	0.088	0.123	0.151	0.184	0.238	0.361
VGG16, last fc	0.082	0.086	0.091	0.094	0.113	0.142	0.193	0.205	0.214

Table 3.16. Kolmogorov-Smirnov 2-sample test p-value quantiles between biases generated by varying the initial random seeds and bootstrapped neural networks. Selected biases match those in Table 3.15.

Layer	**1st**	**5th**	**10th**	**25th**	**50th**	**75th**	**90th**	**95th**	**99th**
CNN3, 1st conv	0.500	0.628	0.647	0.816	0.951	0.984	0.997	0.998	0.999
CNN3, last fc	0.733	0.766	0.807	0.836	0.951	0.976	0.985	0.991	0.996
VGG16, 1st conv	1.1e-8	3.6e-4	8.5e-3	0.052	0.172	0.548	0.845	0.937	0.983
VGG16, last fc	1.5e-3	3.3e-3	5.5e-3	0.092	0.249	0.458	0.497	0.575	0.638

3.8 Regression Model for Estimating Parameters Based on Bootstrap

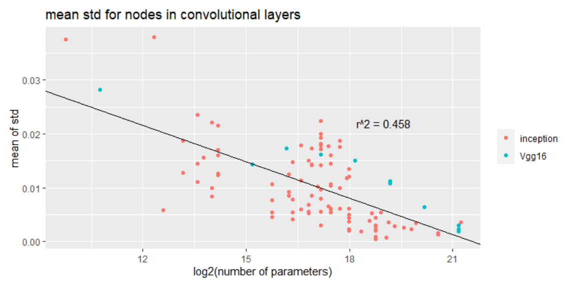
In some instances, it may not be practical or possible to bootstrap the target neural network. Using the distribution information observed in the Bootstrap experiments described in Section 3.3, a method has been created to estimate the distribution parameters in these situations. Rather than estimating node-specific distribution parameters, a single distribution is estimated per layer and is applied to each weight in the layer. Using a regression model, the weight distribution parameters are estimated with number of nodes in a layer being the predictor variable. Regression models were created for both Gaussian and Uniform distributions.

For the Gaussian distribution, the mean is set to the average value of all nodes in a given layer of the network. Variance is then estimated using linear regression. Standard deviations of each weight was taken for the bootstrap experiments run on Inception v3 and VGG-16, described in Sections 3.3.4, 3.3.3. The bootstrap experiments from CNN3 were excluded from the analysis due to the small number of parameters in this neural network. The mean standard deviation was then taken for each layer. Separate regression models were created for convolutional layers and fully connected layers. The formulas for the regression models can be seen in Equations 3.9 and 3.10, where Y is the mean standard deviation of the nodes in the layer and x is the number of nodes in the layer.

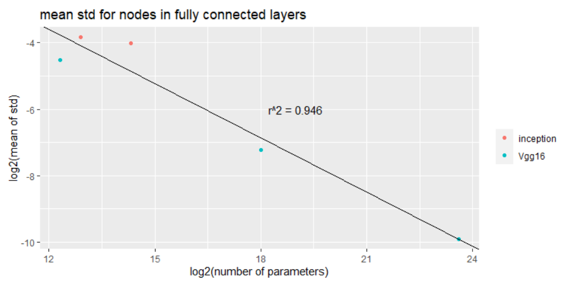
$$Y = 0.048585 - 0.002251 \log_2 x \tag{3.9}$$

$$\log_2 Y = 2.9492 - 0.5451 \log_2 x \tag{3.10}$$

For the Uniform distribution, the interval ranges are estimated for various interval sizes using linear regression. For various interval sizes (e.g., 70% interval, 95% interval) the interval range of each weight was taken for the bootstrap experiments run on Inception v3 and VGG-16, described in Sections 3.3.4, 3.3.3. As above, the bootstrap experiments from CNN3 were excluded from the analysis due to the small number of parameters in this neural network. The mean interval range was then taken for each layer. Separate regression models



(a) Convolutional layers



(b) Fully connected layers

Figure 3.85. Scatterplots of mean standard deviation by number of parameters, including regression line.

were created for convolutional layers and fully connected layers, for each of four interval sizes: 70%, 80%, 90%, and 95%. The formulas for the regression models are summarized in Table 3.17, where Y is the mean interval range of the nodes in the layer and x is the number of nodes in the layer. The upper bound parameter of the Uniform distribution is then set to half of the interval range. The lower bound distribution parameter is the negative of the upper bound.

Table 3.17. Linear Regression results for estimating interval ranges in bootstrap neural networks. Here trans(Y) = β_0 + β_1*log_2 (number of nodes)

Interval size	**Layer type**	**Y transform**	β_0	β_1	**adj** R^2
70	conv	-	0.078455	-0.003597	0.3596
80	conv	-	0.104027	-0.004782	0.3874
90	conv	-	0.14747	-0.006802	0.4234
95	conv	-	0.19050	-0.008812	0.4507
70	fc	log_2	4.3143	-0.5988	0.8312
80	fc	log_2	4.6128	-0.5922	0.864
90	fc	log_2	4.8530	-0.5742	0.9092
95	fc	log_2	4.7749	-0.5420	0.9402

4. RANDOMIZED CNN ENSEMBLE

The goal of bootstrapping the network is to learn distribution information about the nodes in the convolutional layers and the fully connected layer. In particular, we wish to estimate percentiles, mean and variance. Armed with this information, parameters for random distributions can be estimated, and nodes can be randomized according to these distributions. With a portion of the nodes randomized, the individual learner has been weakened. Since even a smaller CNN has a large number of parameters for each layer, replacing some weights by random numbers in a CNN model only slightly reduces the accuracy of the model on clean test data. Then an ensemble of CNN models with some random weights can achieve the same accuracy as the trained model without random weights. A random ensemble also achieves robust performance over adversarial examples.

Given one trained CNN model, we can have infinitely many CNN random ensembles which all maintain the same accuracy over the clean test data as the trained model itself, and achieve robust performance over adversarial examples. We design this strategy to address the adaptive attacks. An adaptive attack will attack the defense strategy itself. By having infinitely many ensembles, we can frequently switch to a different ensemble without incurring additional training time. Thus the adaptive attack loses the target to attack. Although we cannot break the cycle of attacks and defenses, we can manage to stay ahead of the adversaries in the cycle.

4.1 Adversarial images for ensemble

The networks used to implement the ensemble strategy include CNN3, VGG16, Inception v3 and ResNet-18. These network structures are introduced in Chapter 3. The CNN3 architecture will once again be used to test MNIST images, specifically those images which have been adversarially altered. The VGG16 and ResNet18 architectures will again be used to test CIFAR-10 images, both clean and adversarially altered. The Inception v3 architecture will this time be used to test clean and adversarial images generated from the Tiny ImageNet database, which is a subset of the ImageNet database. Tiny ImageNet training data consists of 100,000 images from 200 classes. Each class has 500 training images, 50 test images, and 50 validation images. The images are all 64x64 color images.

The Tiny ImageNet database can be downloaded from ”http://cs231n.stanford.edu/tiny-imagenet-200.zip”. The folder structure is organized into separate folders for train, validation and test images. Within the train folder, there is a folder for each of the 200 classes. Within each class folder, there is an image folder which contains the 500 train images for the class. The test folder contains 10,000 unlabeled images. These images are not used in this work. The validation folder also contains 10,000 unlabeled images. However, there is an annotation text file which names each image and its corresponding class label. In order to get the images into a format which could be used for validating the model, an updated validation folder was created which mimics the structure of the train directory. That is, one folder for each image label, and within that folder, an image folder which contains 50 validation images for that class.

Inception v3 was initialized using the pre-trained weights available on Python. After resetting the fully connected module to have to correct number of output classes, that layer was initialized using Kaiman's uniform method. The Tiny ImageNet images were scaled to 299x299, as required by Torchvision's implementation of Inception v3. This was performed by first resizing the images to 325x325 by padding the images on all sides, then taking the center crop of the required 299x299. During training, a random horizontal flip was implemented, as well as transformation to tensor and normalization. The full database of 100,000 training images was used in retraining. One potential challenge in Tiny ImageNet is that the data size is much larger than MNIST and CIFAR-10. The network was trained with a batch size of 64, but the data was only moved to the GPU in blocks of 100, effectively making batch size jump between 64 and 36. There were 24 epochs of training. Optimization was performed with SGD, with momentum of 0.9 and weight decay 0.0005. The initial learning rate was 0.05 and decayed by a factor of 0.5 every 30 epochs. The final model used for the random ensemble has Top-1 accuracy of 80.0% on 10,000 Tiny ImageNet validation images.

4.2 Adversarial images for ensemble

For each of the four networks, a set of 4000 adversarial examples was generated. Four white-box attack methods were implemented, with 1000 examples generated from each. Each attack generated adversarial examples based on the same 1000 clean images, with an even distribution of examples from each class from the original datasets. This amounts to 100 images per class for MNIST and CIFAR10 databases, and 5 images per class for Tiny ImageNet. The attacks used were Fast gradient sign method (FGSM), Projected gradient descent (PGD), DeepFool (DP), and Carlini-Wagner L2 (C&W). Details of these methods are discussed in Section 2.1.1. Adversarial examples were generated using the 'foolbox' Python library. Attack accuracies are summarized in Table 4.1.

For the FGSM and PGD attacks, ϵ is a hyperparameter which dictates how far the adversarial example can be from the clean image. For the one-step FGSM attack, it is the only hyperparameter. PGD is an iterative FGSM which stays within a ϵ-neighborhood of the original image, and has many starting points. It has hyperparameters controlling the number of steps and the relative size of each step compared with ϵ. DP uses projections onto a hyperplane of possible classes and has hyperparameters of number of iterations, and the number of candidate projections, or the number of candidate adversarial classes. It also has a hyperparameter dictating how far to overshoot the boundary. C&W's L2 attack solves a box constraint optimization problem, with hyperparameters for the initialization of the training constant and number of binary search steps, number of steps within each binary step, step size, confidence with which classification occurs. Although an ϵ may be enforced when searching for DP and C&W solutions, it has no effect on the decisions being made by the algorithms.

The baseline CNN3 model achieved 99.4% accuracy on the 1000 clean MNIST images prior to introdution of adversarial perturbation. The PGD attack decreased the accuracy to 0.8% on these same images, with an L2-distance of 10. The accuracies, average L2-distances and training hyperparameters of the adversarial examples from all attack types can be viewed in the following table.

The baseline VGG16 and ResNet-18 models achieved 93.7% and 88.7% accuracy on the 1000 clean CIFAR-10 images prior to introduction of adversarial perturbation. The C&W attack on ResNet decreased the accuracy to 8.5% on these same images, with an average L2-distance of 0.91. This same attack decreased accuracy on VGG16 to 1.5% with an average L2-distance of 0.83. Full attack details are available below.

The baseline Inception v3 models achieved 80.0% accuracy on the 1000 clean Tiny ImageNet images prior to introduction of adversarial perturbation. The DP attack decreased the accuracy to 3.3% on these same images, with an average L2-distance of 0.72. As the number of candidate classes was set to 10 and this is the only network with more than 10 classes, this is the only network example which utilized the computation decreasing feature available in the DP implementation. It can be seen that it did not negatively effect the overall performance of the attack algorithm. Full attack details for Inception v3 are available below.

Table 4.1. Accuracy of undefended neural networks on clean and adversarial images

Model	**Clean images**	**PGD**	**FGSM**	**DeepFool**	**C&W2**
CNN3	0.991	0.029	0.082	0.148	0.275
VGG16	0.889	0.013	0.043	0.067	0.016
ResNet18	0.834	0.032	0.027	0.053	0.081
Incept v3	0.773	0.079	0.203	0.075	0.137

4.3 Randomization

Ensembles are generated from a single fully trained network. One layer of the architecture is selected to be randomized. For ensemble size n, there are n copies of the randomized layer. The reminder of the parameters in the network remain unchanged. For each of the n versions of the selected layer, each weight in the layer is updated to a random number with a predetermined probability, p. The decision to update a specific node is made using a Bernoulli distribution. The selected layer and the update rate, p, remain constant for all learners in the ensemble.

The random number to which nodes are updated follow a distribution with parameters learned from bootstrap. CNN3 trained on MNIST and VGG16 trained on CIFAR-10 were used for both bootstrap and generating ensembles. For these networks, the updated weight distribution for the ensemble will be based directly on the information from bootstrap. Ensembles with Gaussian distributed weights and ensembles with Uniformly distributed weights are both included. A univariate Gaussian is used, with the mean being equal to the mean of the bootstrap parameter means. The standard deviation is the mean of the bootstrap parameter standard deviations. Several Uniform distributions are used. The Uniform distribution is zero-centered, with the upper and lower bounds equal to +/-range, where range is the distance between 97.5th percentile and the 2.5th percentile, corresponding to the middle 95% of the data. Ensembles were also created with Uniform distributions with parameters estimated from the middle 90%, middle 80%, etc. The middle ranges were estimated per parameter via bootstrap and then the average was taken for the layer.

We also generate ensembles for ResNet-18 trained on CIFAR-10 and Inception v3 trained on Tiny ImageNet. For these networks, rather than using bootstrap information directly, we infer from other bootstrap networks and design a linear model to estimate the distribution parameter values. Details for the linear regression are given in Section 3.8. Univariate Gaussian and Uniform distributions are used as with the other networks. However, rather than using an estimate for the mean, the mean is set to the average weight observed in the layer.

Preliminary results indicated that randomization of the earliest convolutional layers demonstrated a greater robustness than randomizing later convolutional layers or fully connected layers. All ensembles presented here have layers selected from the beginning of the network. For each layer presented here, factorial experiments were run for all values p = 0.1, 0.2, ... 0.9 and distributions based on standard deviation and middle range = 70%, 80%, 90% and 95%. Each design setting was repeated 5 times. Sometimes additional values of p and additional middle range values were also included.

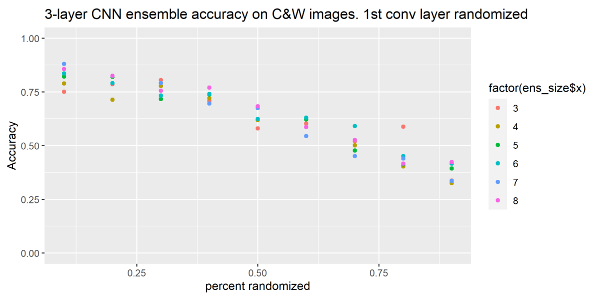
4.4 Testing the ensembles

For each input image, a neural network produces a probability of the input belonging to each class. The output is averaged across all networks in the ensemble, and the assigned class corresponds to the class index with the maximum value.

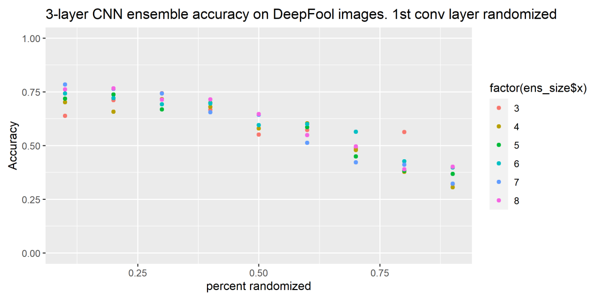
Effect of L2-distance on ensemble performance. A design choice was made to prioritize adversarial examples with low L2 rather than maximizing the attack success. In a real world setting, where L2-distance of an adversarial image is unknown, this step could be estimated by utilizing a 'selective prediction' algorithm to detect out of distribution samples. Examples of these techniques can be found in [39] and Section 2.1.3.

4.5 Plots with normal randomization

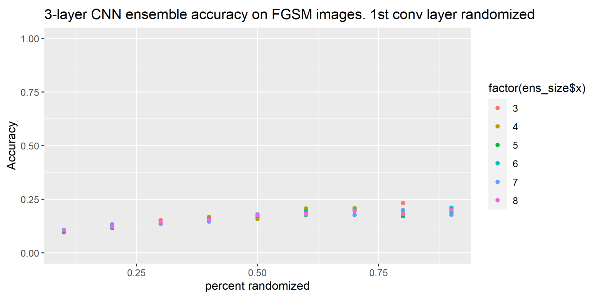
Parameters for the Gaussian distributions were determined using the method described in Section 4.3. We achieve the best performance on randomizing Inception v3 which has the most complex model structure. We randomize only one layer for building the ensemble. We discover that using random weights at the front of the model achieves the best results on correctly classify the adversarial images and maintain the accuracy on the clean data. For smaller DNN models, the performance of the random ensemble on the adversarial images with larger perturbations start to drop. Every accuracy result reported here is based on the average of 5 runs with the same setup.



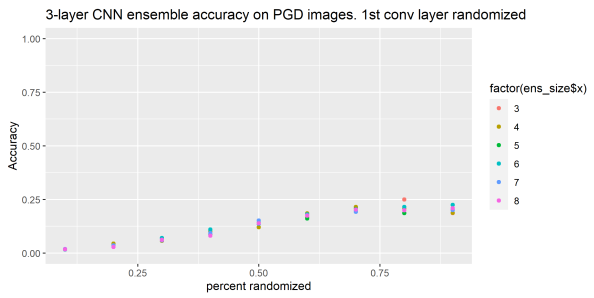
(a) CW attack



(b) DP attack



(c) FGSM attack

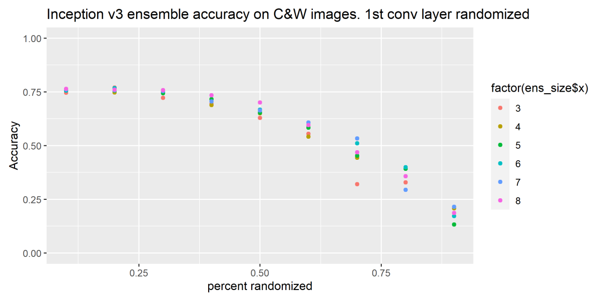


(d) PGD attack

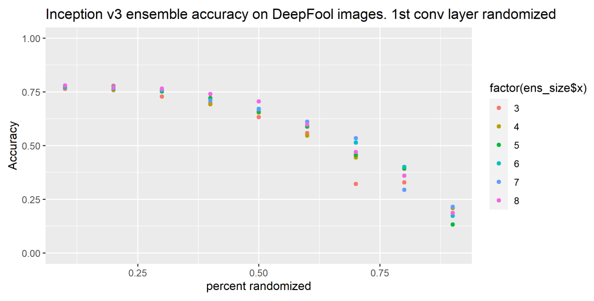
Figure 4.1. Ensemble results for Gaussian randomization of CNN3, first convolutional layer

Figure 4.1 shows ensemble results for the first convolutional layer of CNN3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the C&W attack and DeepFool attack, randomizing 10% of the weights of the convolutional layer achieves the highest accuracy. For both attacks, ensemble size 7 achieves the highest accuracy of 88.0% and 78.4% respectively. For FGSM and PGD attacks, randomizing 80% of the weights of the convolutional layer achieves the highest accuracy. For FGSM and PGD, ensemble size 3 has the highest accuracy of 23.1% and 24.9%.

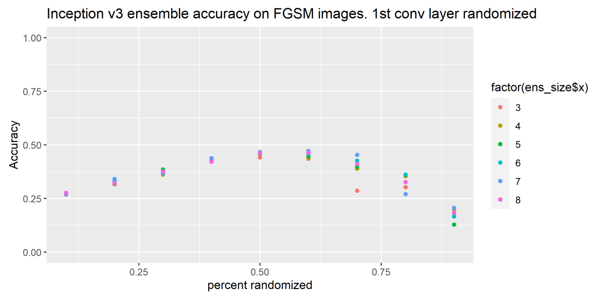
Figure 4.2 shows ensemble results for the first convolutional layer of Inception v3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the DeepFool attack, randomizing 10% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 7 achieves the highest accuracy of 78.0%. For the C&W attack, randomizing 20% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 3 achieves the highest accuracy of 77.0%.



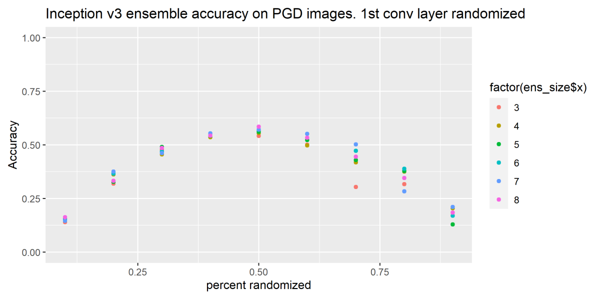
(a) CW attack



(b) DP attack



(c) FGSM attack



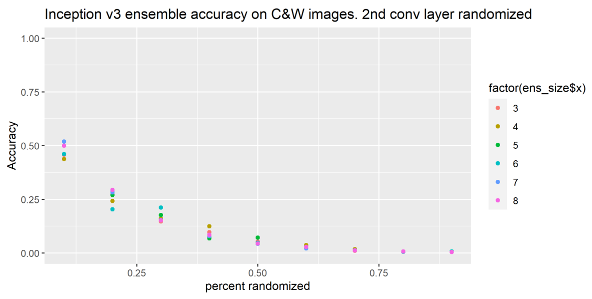
(d) PGD attack

Figure 4.2. Ensemble results for Gaussian randomization of Incept v3, first convolutional layer

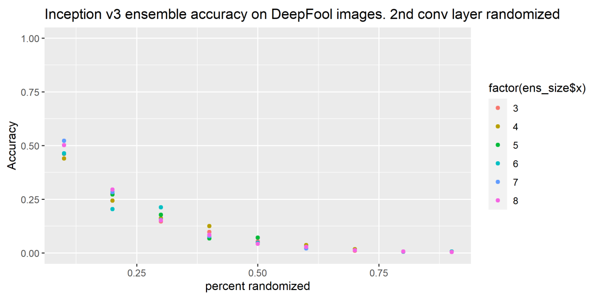
For FGSM, randomizing 60% of weights with ensemble size 7 achieves accuracy 47.2%. For PGD, randomizing 50% of weights with ensemble size 8 achieves accuracy 58.5%.

Figure 4.3 shows ensemble results for the second convolutional layer of Inception v3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the DeepFool attack, randomizing 1% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 7 achieves the highest accuracy of 77.4%. For the C&W attack, randomizing 1% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 5 achieves the highest accuracy of 76.3%. For FGSM and PGD attacks, randomizing 5% of weights with ensemble size 8 achieves the highest accuracy of 43.7% and 57.7%, respectively.

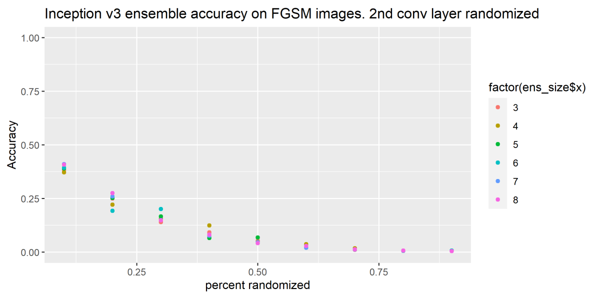
Figure 4.4 shows ensemble results for the third convolutional layer of Inception v3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the C&W and DeepFool attacks, randomizing 10% of the



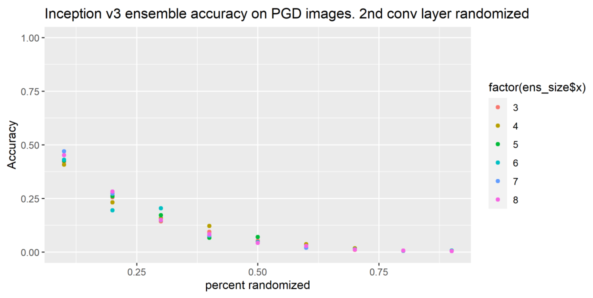
(a) CW attack



(b) DP attack



(c) FGSM attack



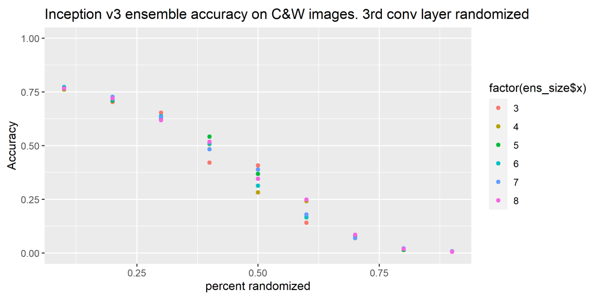
(d) PGD attack

Figure 4.3. Ensemble results for Gaussian randomization of Incept v3, second convolutional layer

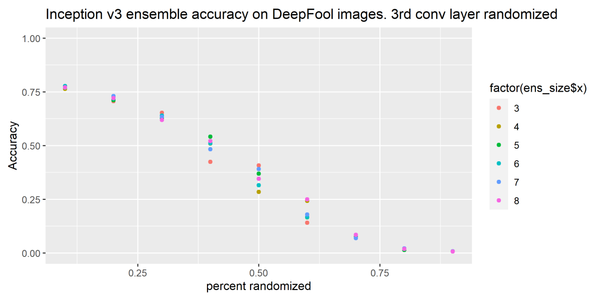
weights of the convolutional layer achieves the highest accuracy. For both attacks, ensemble size 6 achieves the highest accuracy of 77.2% and 77.7%, respectively. For FGSM, randomizing 30% of weights with ensemble size 3 achieves accuracy 49.7%. For PGD, randomizing 20% of weights with ensemble size 7 achieves accuracy 60.4%.

Figure 4.5 shows ensemble results for the fourth convolutional layer of Inception v3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the C&W and DeepFool attacks, randomizing 10% of the weights of the convolutional layer achieves the highest accuracy. For both attacks, ensemble size 8 achieves the highest accuracy of 74.1% and 75.1%, respectively. For FGSM, randomizing 30% of weights with ensemble size 6 achieves accuracy 42.4%. For PGD, randomizing 20% of weights with ensemble size 7 achieves accuracy 50.0%.

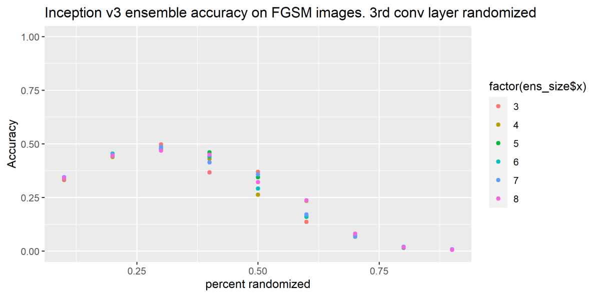
Figure 4.6 shows ensemble results for the fifth convolutional layer of Inception v3, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging



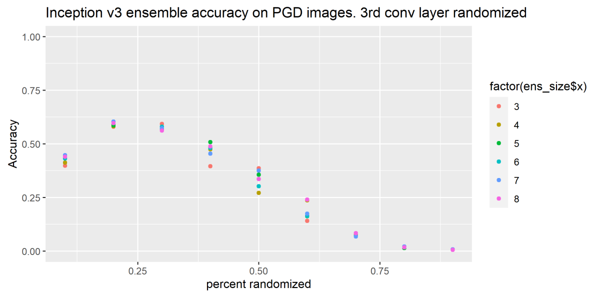
(a) CW attack



(b) DP attack



(c) FGSM attack

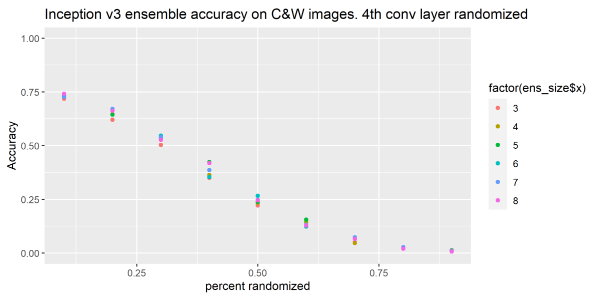


(d) PGD attack

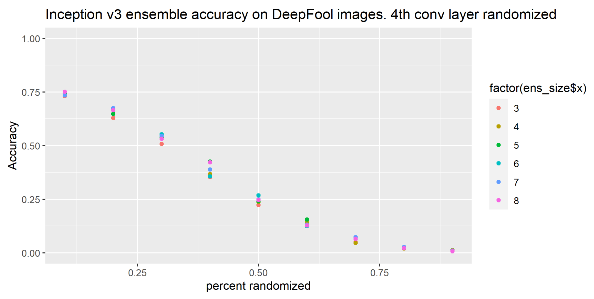
Figure 4.4. Ensemble results for Gaussian randomization of Incept v3, third convolutional layer

from 3 to 8 randomized models. For the C&W and DeepFool attacks, randomizing 10% of the weights of the convolutional layer achieves the highest accuracy. For both attacks, ensemble size 8 achieves the highest accuracy of 74.9% and 76.2%, respectively. For FGSM, randomizing 30% of weights with ensemble size 8 achieves accuracy 41.7%. For PGD, randomizing 30% of weights with ensemble size 7 achieves accuracy 48.2%.

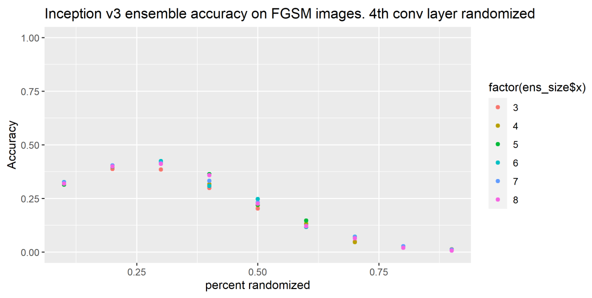
Figure 4.7 shows ensemble results for the first convolutional layer of ResNet-18, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the C&W attack, randomizing 5% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 6 achieves the highest accuracy of 75.6%. For DeepFool, randomizing 10% of weights with ensemble size 8 achieves accuracy 62.7%. For FGSM, randomizing 30% of weights with ensemble size 7 achieves accuracy 29.7%. For PGD, randomizing 40% of weights with ensemble size 4 achieves accuracy 36.0%.



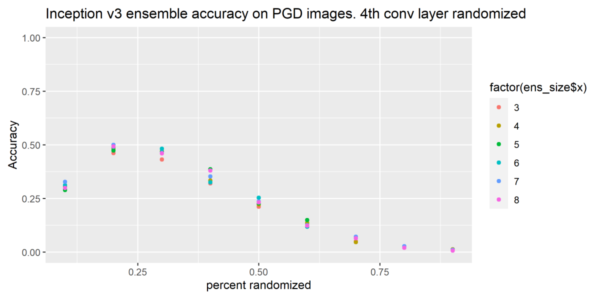
(a) CW attack



(b) DP attack



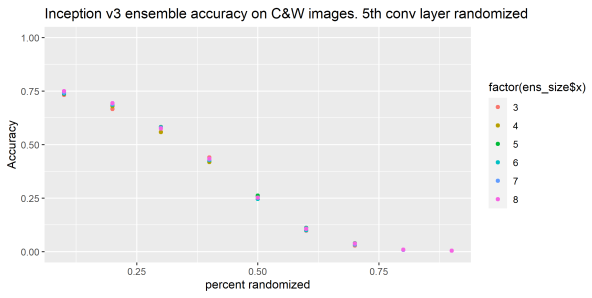
(c) FGSM attack



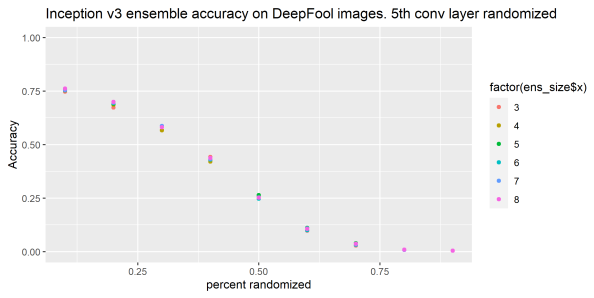
(d) PGD attack

Figure 4.5. Ensemble results for Gaussian randomization of Incept v3, fourth convolutional layer

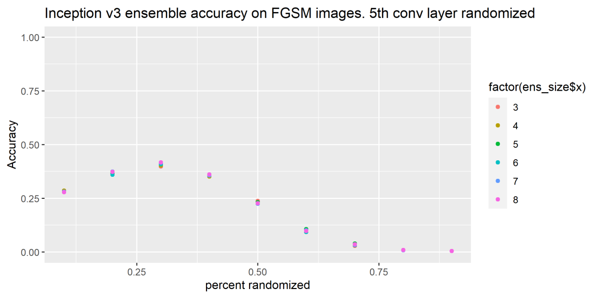
Figure 4.8 shows ensemble results for the first convolutional layer of VGG16, using Gaussian randomization. It includes four adversarial attacks, and ensemble sizes ranging from 3 to 8 randomized models. For the C&W attack, randomizing 10% of the weights of the convolutional layer achieves the highest accuracy. Ensemble size 8 achieves the highest accuracy of 76.4%. For DeepFool, randomizing 30% of weights with ensemble size 5 achieves accuracy 45.6%. For FGSM, randomizing 70% of weights with ensemble size 3 achieves accuracy 18.9%. For PGD, randomizing 60% of weights with ensemble size 8 achieves accuracy 33.7%.



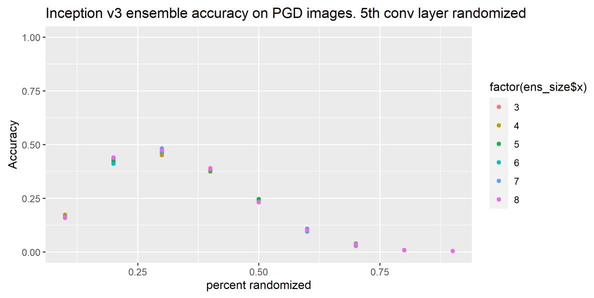
(a) CW attack



(b) DP attack

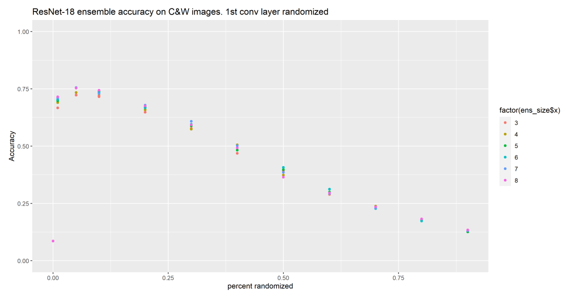


(c) FGSM attack

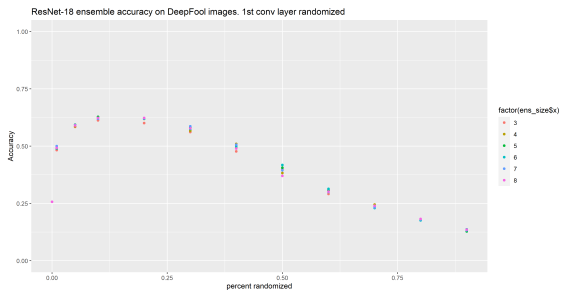


(d) PGD attack

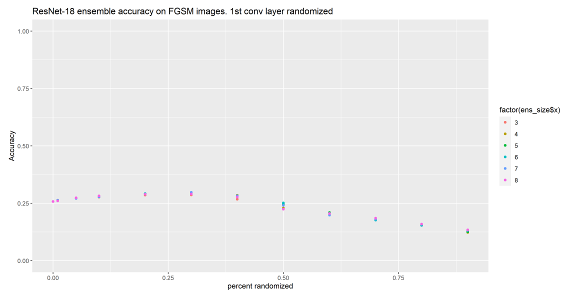
Figure 4.6. Ensemble results for Gaussian randomization of Incept v3, fifth convolutional layer



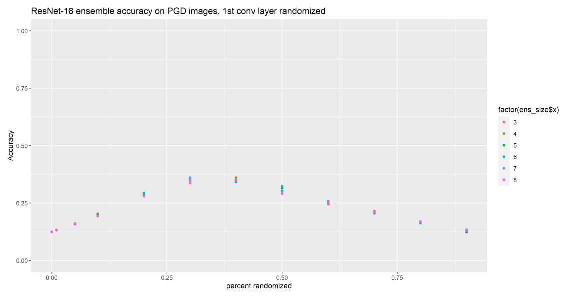
(a) CW attack



(b) DP attack

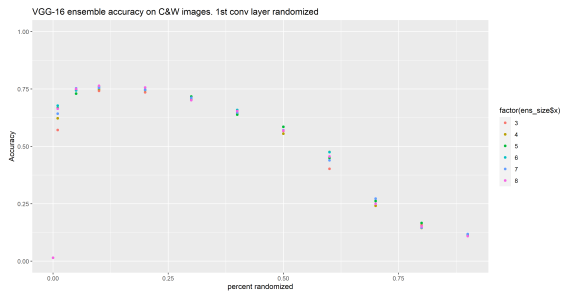


(c) FGSM attack

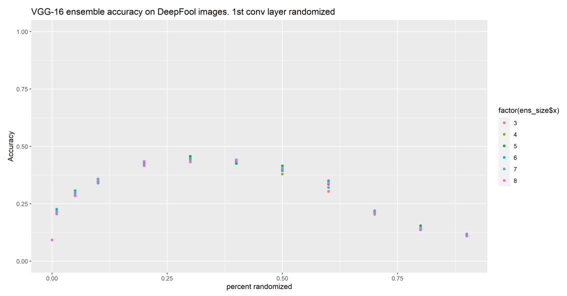


(d) PGD attack

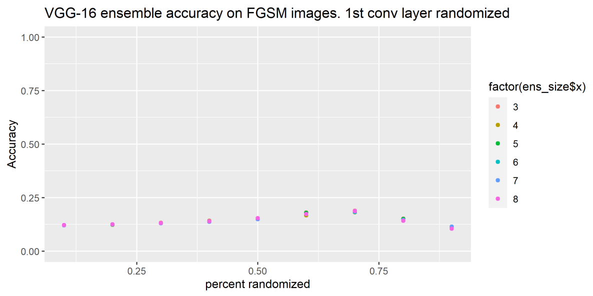
Figure 4.7. Ensemble results for Gaussian randomization of ResNet-18, first convolutional layer



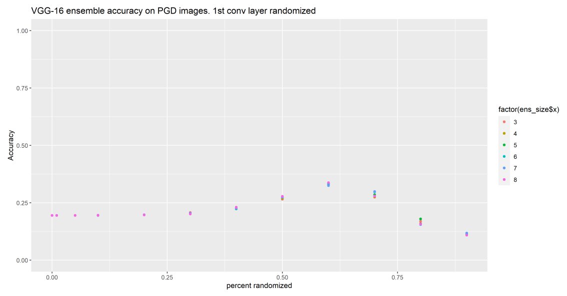
(a) CW attack



(b) DP attack



(c) FGSM attack



(d) PGD attack

Figure 4.8. Ensemble results for Gaussian randomization of VGG-16, first convolutional layer

4.6 Plots with uniform randomization

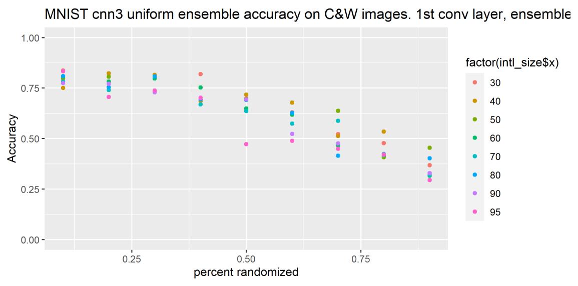
For CNN3 trained on MNIST and VGG16 trained on CIFAR-10, the weight distribution for the ensemble will be based directly on the information from bootstrap. Several Uniform distributions are used. The Uniform distribution is zero-centered, with the upper and lower bounds equal to +/-range, where range is the distance between 97.5th percentile and the 2.5th percentile, corresponding to the middle 95% of the data. Ensembles were also created with Uniform distributions with parameters estimated from the middle 90%, middle 80%, etc. The middle ranges were estimated per parameter via bootstrap and then the average was taken for the layer.

We also generate ensembles for ResNet-18 trained on CIFAR-10 and Inception v3 trained on Tiny ImageNet. For these networks, rather than using bootstrap information directly, we infer from other bootstrap networks and design a linear model to estimate the distribution parameter values. Details for the linear regression are given in Section 3.8.

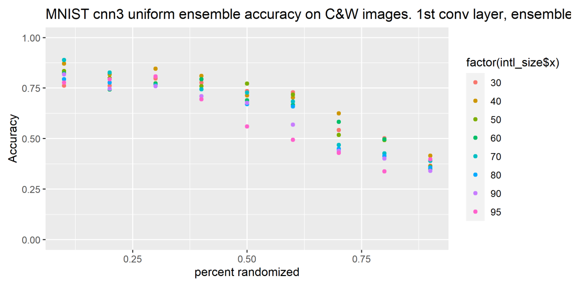
Again we achieve the best performance on randomizing Inception v3 using uniform random weights which has the most complex model structure. We randomize only one layer for building the ensemble. For smaller DNN models, the performance of the random ensemble on the adversarial images with larger perturbations start to drop. Every accuracy result reported here is based on the average of 5 runs with the same setup.

In Figure 4.9, for the CW attack, for all ensemble sizes, 10% random weights on the first convolutional layer achieves the highest accuracy. Ensemble size 3 has the highest accuracy 83.7%, with 30% interval. Ensemble size 4 has the highest accuracy 88.8% with 70% interval. Ensemble size 5 has the highest accuracy 88.6% with 70% interval. Ensemble size 6 has the highest accuracy 93.6% with 30% interval. Ensemble size 7 has the highest accuracy 88.0% with 40% interval. Ensemble size 8 has the highest accuracy 89.9% with and 40% interval.

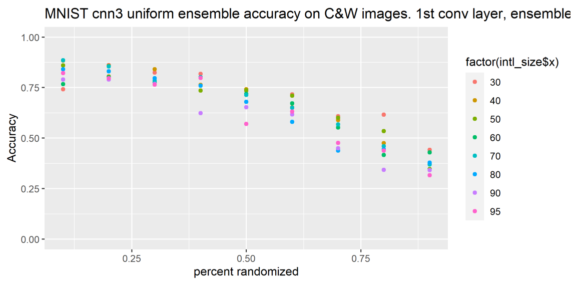
In Figure 4.10, we see accuracy on the DeepFool attack images with CNN3 Uniform ensembles. Ensemble size 3 has the highest accuracy 74.4%, with 10% of weights in the first convolutional layer randomized and 95% interval. Ensemble size 4 has the highest accuracy 77.7%, with 10% of weights in the first convolutional layer randomized and 70% interval. Ensemble size 5 has the highest accuracy 78.1%, with 20% of weights in the first convolutional



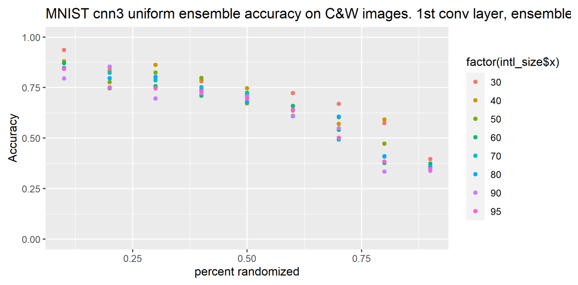
(a) Ensemble size 3



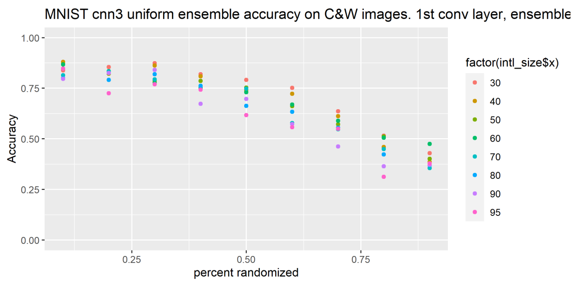
(b) Ensemble size 4



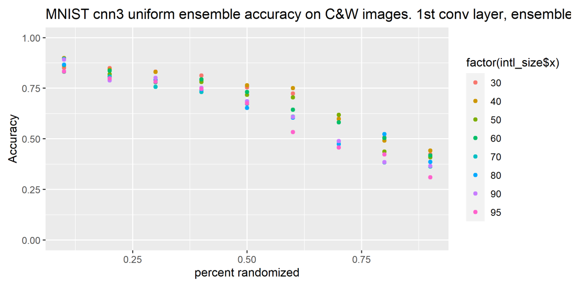
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

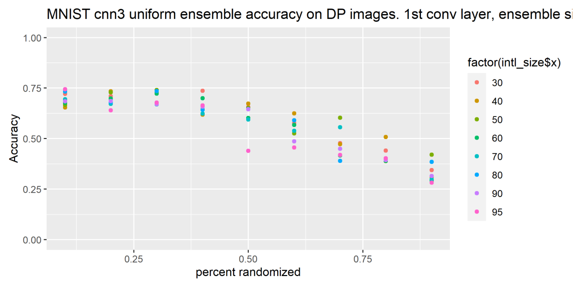


(f) Ensemble size 8

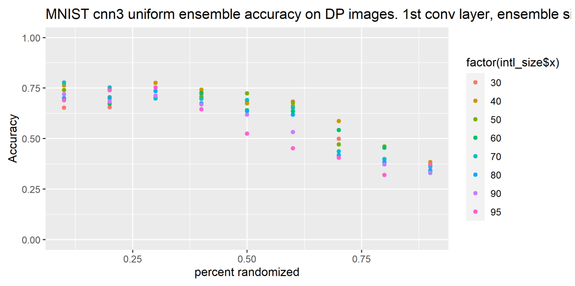
Figure 4.9. Ensemble results for Uniform randomization of CNN3, first convolutional layer, on C&W attack images

layer randomized and 30% interval. Ensemble size 6 has the highest accuracy 84.2%, with 10% of weights in the first convolutional layer randomized and 30% interval. Ensemble size 7 has the highest accuracy 79.2%, with 30% of weights in the first convolutional layer randomized and 40% interval. Ensemble size 8 has the highest accuracy 80.1%, with 10% of weights in the first convolutional layer randomized and 40% interval.

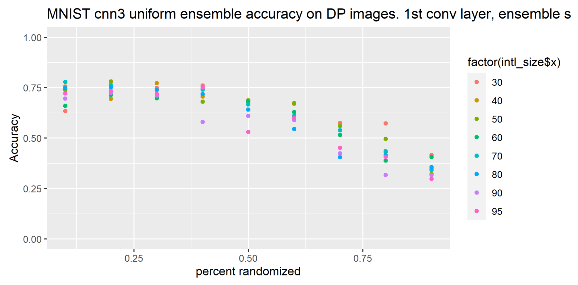
In Figure 4.11, we see accuracy on the FGSM attack images with CNN3 Uniform ensembles. Ensemble size 3 has the highest accuracy 19.7%, with 70% of weights in the first



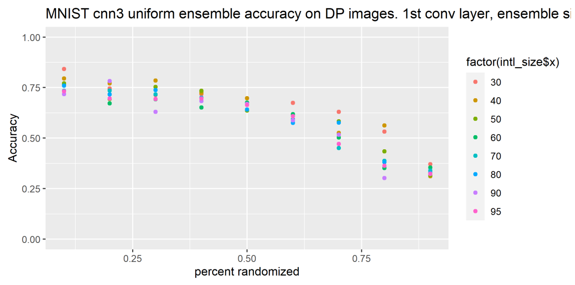
(a) Ensemble size 3



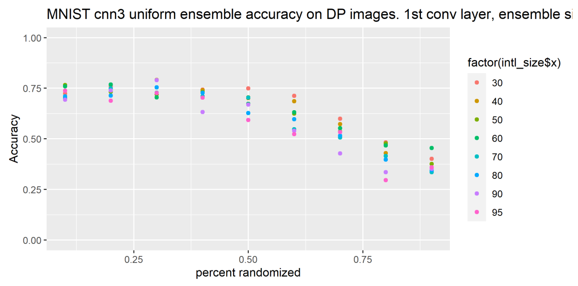
(b) Ensemble size 4



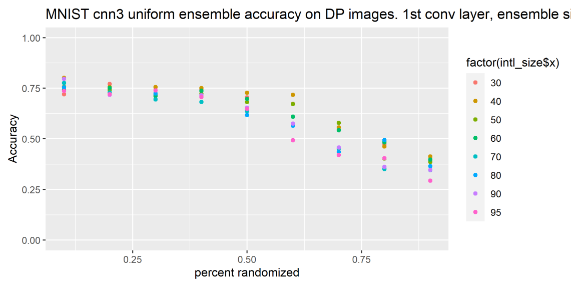
(c) Ensemble size 5



(d) Ensemble size 6



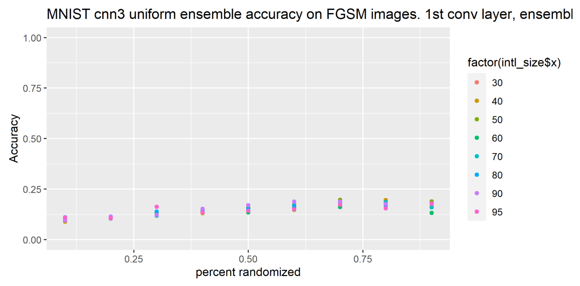
(e) Ensemble size 7



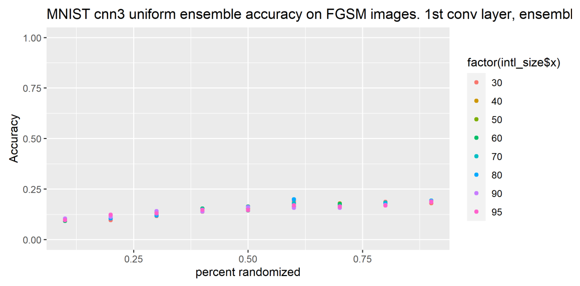
(f) Ensemble size 8

Figure 4.10. Ensemble results for Uniform randomization of CNN3, first convolutional layer, on DeepFool attack images

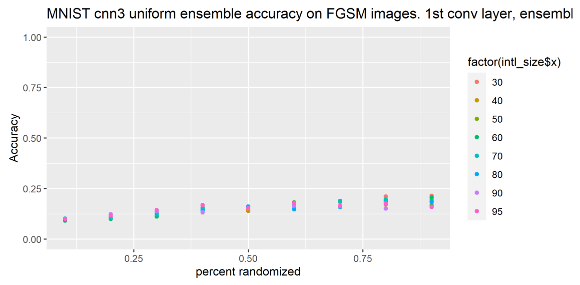
convolutional layer randomized and 50% interval. Ensemble size 4 has the highest accuracy 19.9%, with 60% of weights in the first convolutional layer randomized and 80% interval. Ensemble size 5 has the highest accuracy 21.5%, with 90% of weights in the first convolutional layer randomized and 30% interval. Ensemble size 6 has the highest accuracy 20.2%, with 80% of weights in the first convolutional layer randomized and 30% interval. Ensemble size 7 has the highest accuracy 21.7%, with 90% of weights in the first convolutional layer



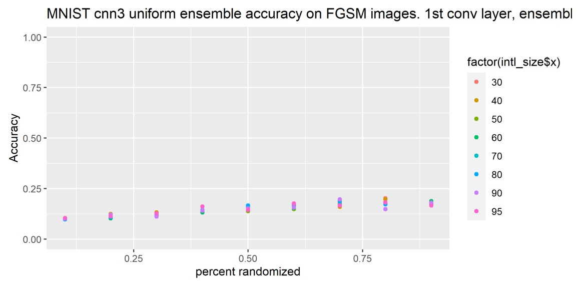
(a) Ensemble size 3



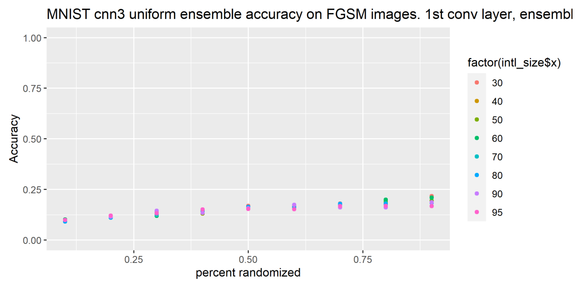
(b) Ensemble size 4



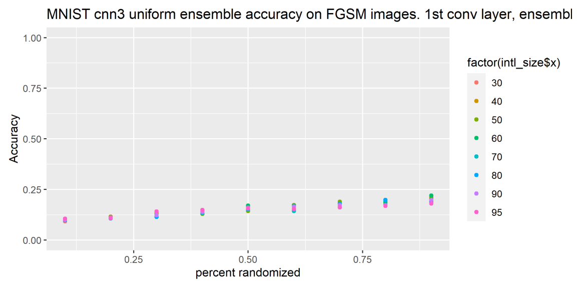
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

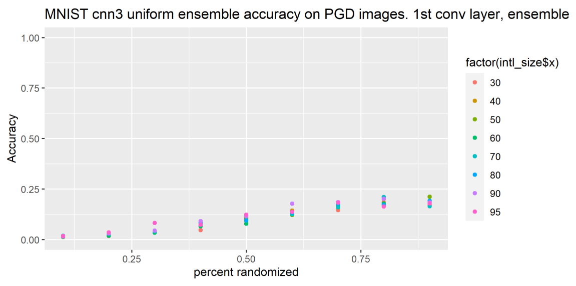


(f) Ensemble size 8

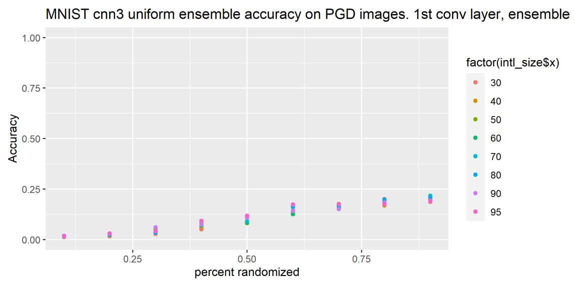
Figure 4.11. Ensemble results for Uniform randomization of CNN3, first convolutional layer, on FGSM attack images

randomized and 30% interval. Ensemble size 8 has the highest accuracy 22.0%, with 90% of weights in the first convolutional layer randomized and 60% interval.

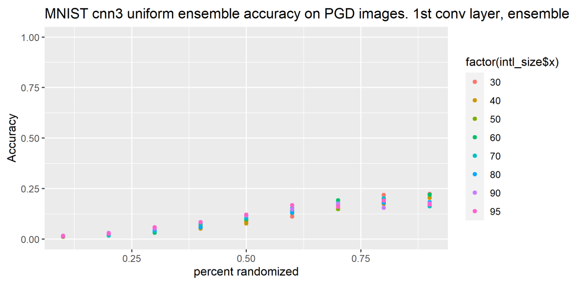
In Figure 4.12, we see accuracy on the PGD attack images with CNN3 Uniform ensembles. Ensemble size 3 has the highest accuracy 21.2%, with 90% of weights in the first convolutional layer randomized and 50% interval. Ensemble size 4 has the highest accuracy 21.8%, with 90% of weights in the first convolutional layer randomized and 70% interval. Ensemble size 5 has the highest accuracy 22.3%, with 90% of weights in the first convolutional layer



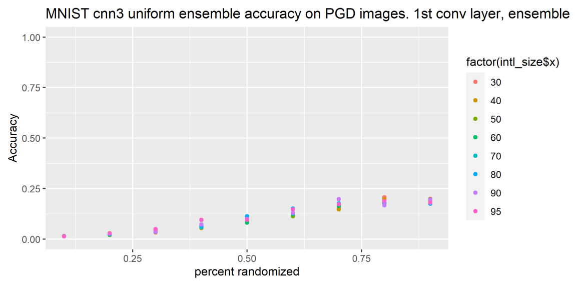
(a) Ensemble size 3



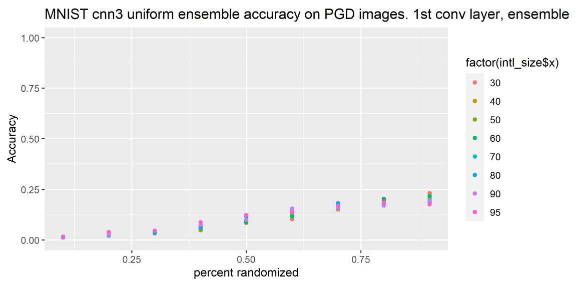
(b) Ensemble size 4



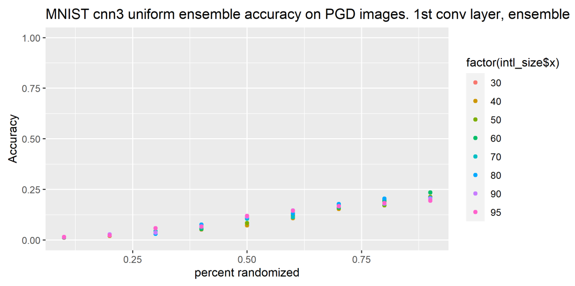
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

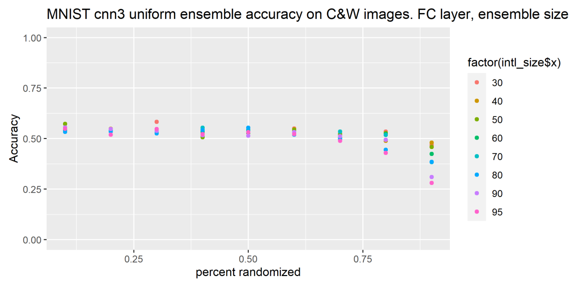


(f) Ensemble size 8

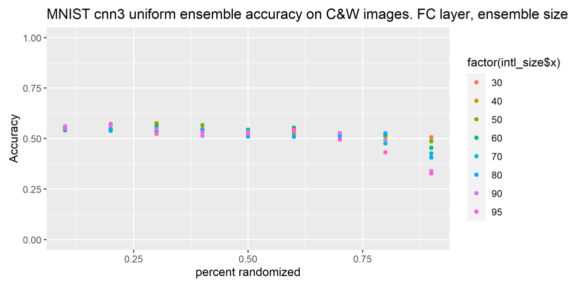
Figure 4.12. Ensemble results for Uniform randomization of CNN3, first convolutional layer, on PGD attack images

randomized and 30% interval. Ensemble size 6 has the highest accuracy 20.7%, with 80% of weights in the first convolutional layer randomized and 30% interval. Ensemble size 7 has the highest accuracy 23.1%, with 90% of weights in the first convolutional layer randomized and 30% interval. Ensemble size 8 has the highest accuracy 23.5%, with 90% of weights in the first convolutional layer randomized and 60% interval.

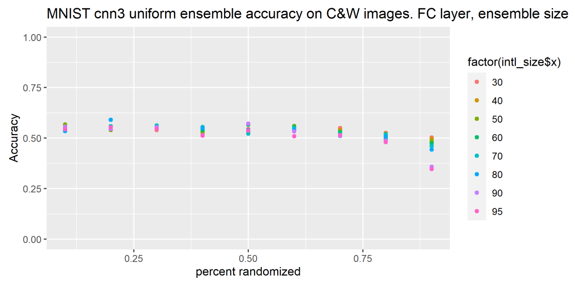
In Figure 4.13, we see accuracy on the C&W attack images with CNN3 Uniform ensembles which randomize the fully connected layer. Ensemble size 3 has the highest accuracy 58.3%,



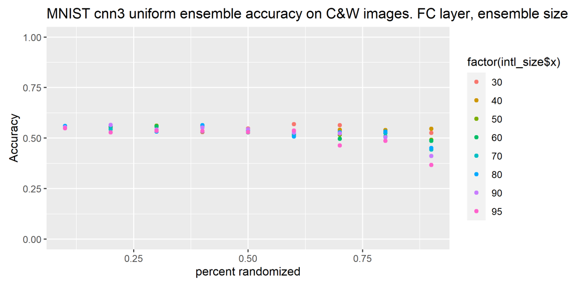
(a) Ensemble size 3



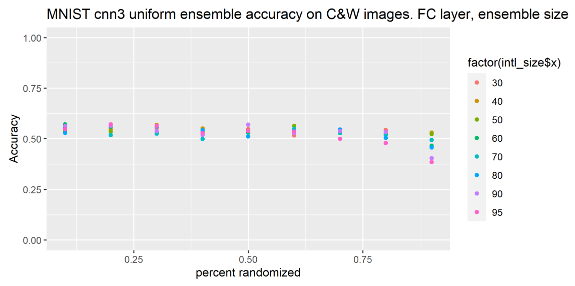
(b) Ensemble size 4



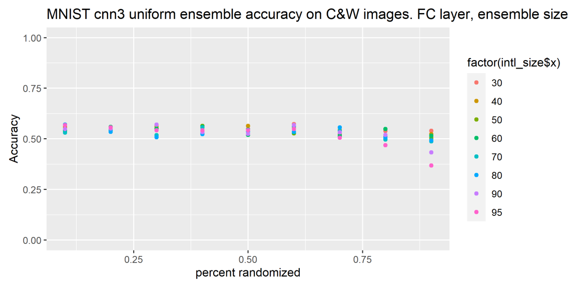
(c) Ensemble size 5



(d) Ensemble size 6



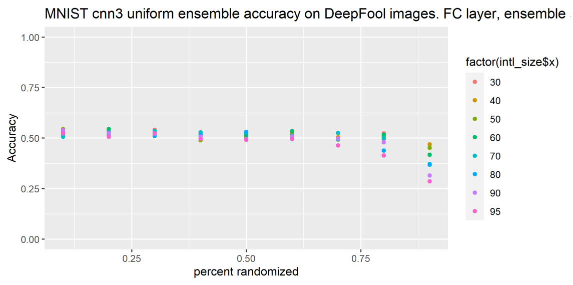
(e) Ensemble size 7



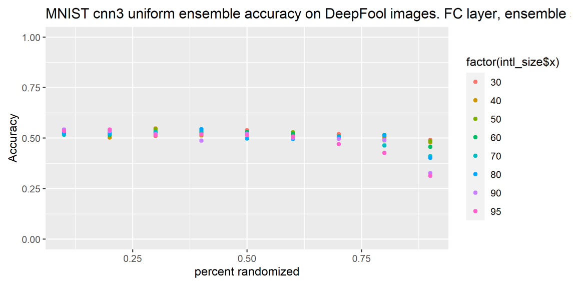
(f) Ensemble size 8

Figure 4.13. Ensemble results for Uniform randomization of CNN3, fully connected layer, on C&W attack images

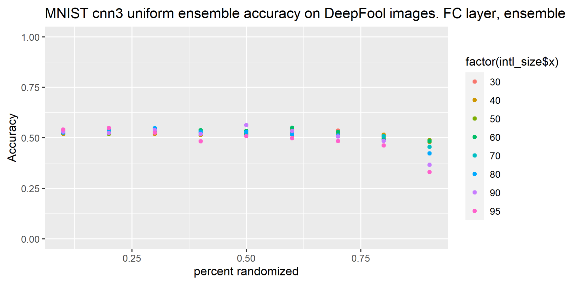
with 30% of weights in the fully connected layer randomized and 30% interval. Ensemble size 4 has the highest accuracy 57.7%, with 30% of weights in the fully connected layer randomized and 40% interval. Ensemble size 5 has the highest accuracy 59.0%, with 20% of weights in the fully connected layer randomized and 80% interval. Ensemble size 6 has the highest accuracy 56.9%, with 60% of weights in the fully connected layer randomized and 30% interval. Ensemble size 7 has the highest accuracy 57.2%, with 10% of weights in the fully connected layer randomized and 60% interval. Ensemble size 8 has the highest accuracy 57.3%, with 60% of weights in the fully connected layer randomized and 30% interval.



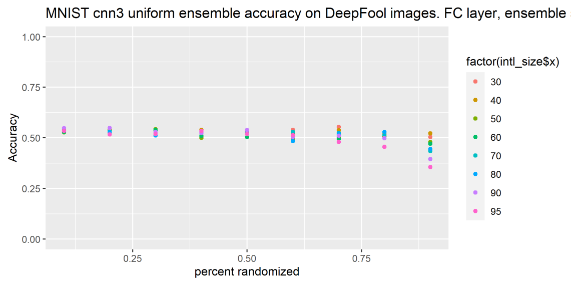
(a) Ensemble size 3



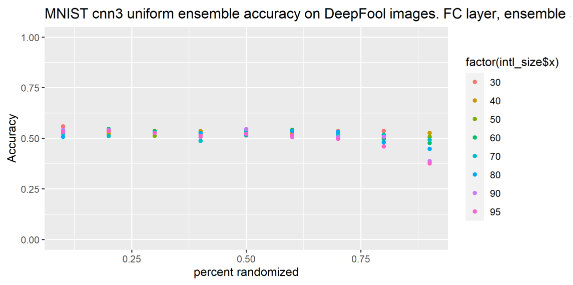
(b) Ensemble size 4



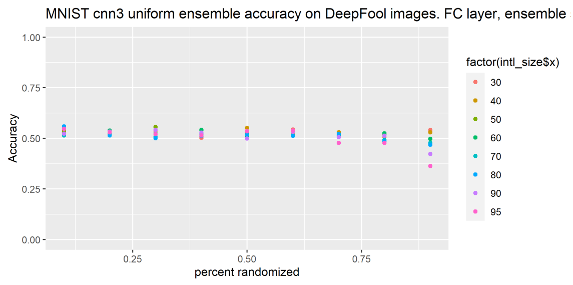
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

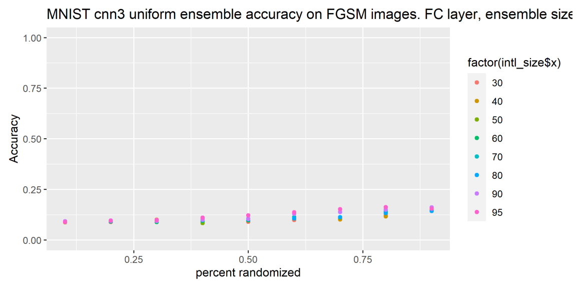


(f) Ensemble size 8

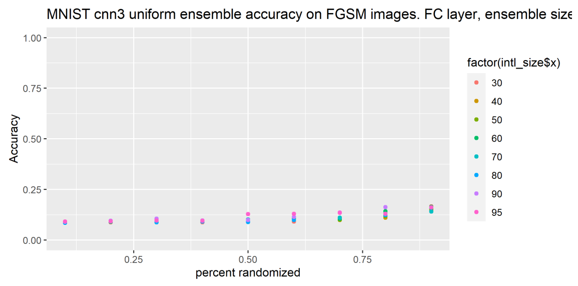
Figure 4.14. Ensemble results for Uniform randomization of CNN3, fully connected layer, on DeepFool attack images

In Figure 4.14, we see accuracy on the DeepFool attack images with CNN3 Uniform ensembles which randomize the fully connected layer. Ensemble size 3 has the highest accuracy 54.5%, with 20% of weights in the fully connected layer randomized and 60% interval. Ensemble size 4 has the highest accuracy 54.7%, with 30% of weights in the fully connected layer randomized and 40% interval. Ensemble size 5 has the highest accuracy 56.2%, with

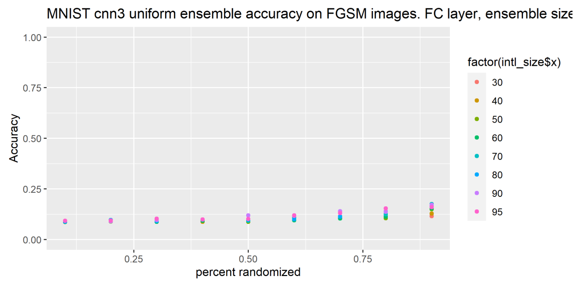
50% of weights in the fully connected layer randomized and 90% interval. Ensemble size 6 has the highest accuracy 55.3%, with 70% of weights in the fully connected layer randomized and 30% interval. Ensemble size 7 has the highest accuracy 55.9%, with 10% of weights in the fully connected layer randomized and 30% interval. Ensemble size 8 has the highest accuracy 55.9%, with 10% of weights in the fully connected layer randomized and 80% interval.



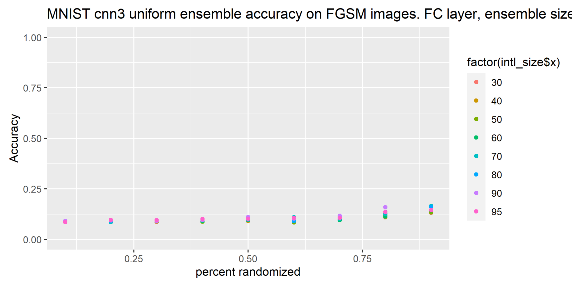
(a) Ensemble size 3



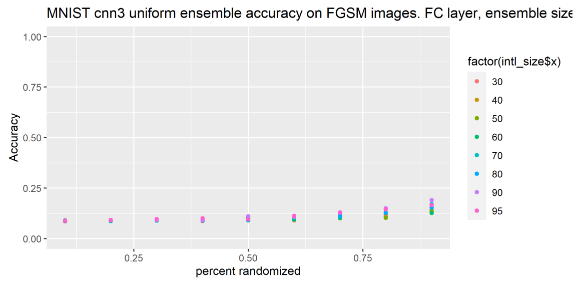
(b) Ensemble size 4



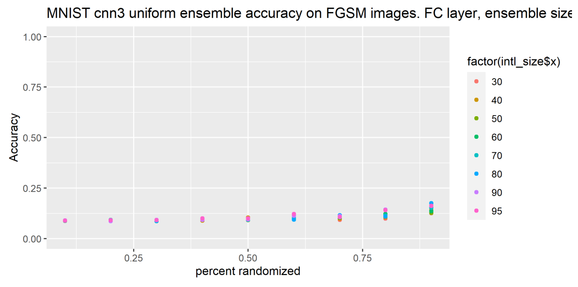
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

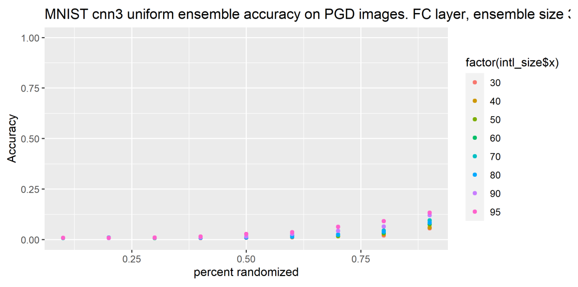


(f) Ensemble size 8

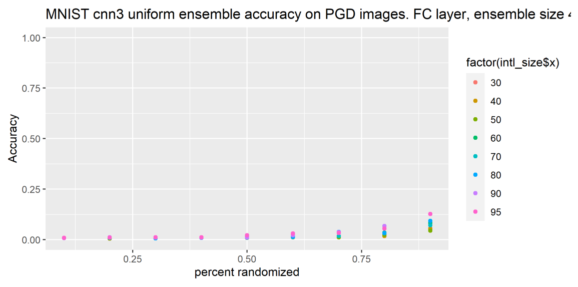
Figure 4.15. Ensemble results for Uniform randomization of CNN3, fully connected layer, on FGSM attack images

In Figure 4.15, we see accuracy on the FGSM attack images with CNN3 Uniform ensembles which randomize the fully connected layer. Ensemble size 3 has the highest accuracy 16.3%, with 80% of weights in the fully connected layer randomized and 95% interval. Ensemble size 4 has the highest accuracy 16.6%, with 90% of weights in the fully connected layer randomized and 30% interval. Ensemble size 5 has the highest accuracy 17.5%, with 90% of weights in the fully connected layer randomized and 70% interval. Ensemble size 6 has the highest accuracy 16.6%, with 90% of weights in the fully connected layer randomized and 70% interval. Ensemble size 7 has the highest accuracy 19.0%, with 90% of weights in the fully connected layer randomized and 90% interval. Ensemble size 8 has the highest accuracy 17.6%, with 90% of weights in the fully connected layer randomized and 80% interval.

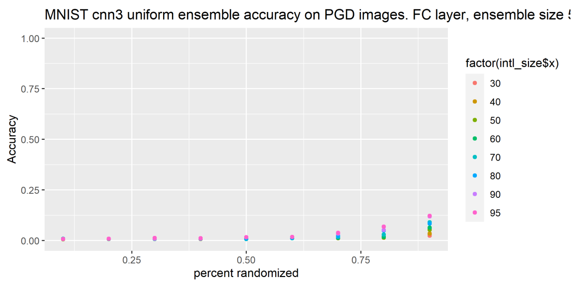
In Figure 4.16, we see accuracy on the PGD attack images with CNN3 Uniform ensembles which randomize the fully connected layer. For all ensemble sizes, the highest accuracies were achieved with 90% of weights in the fully connected layer randomized, with randomization to 95% interval size. Highest accuracies for ensembles sizes 3 to 8 are 13.3%, 12.7%, 12.2%, 9.8%, 12.2%, and 11.3%.



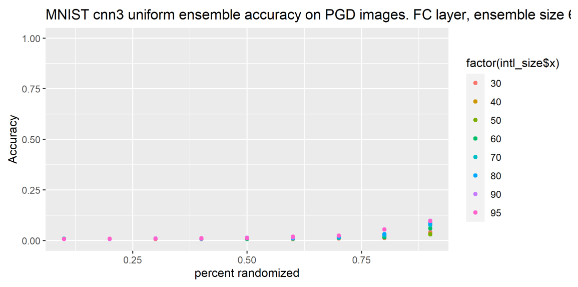
(a) Ensemble size 3



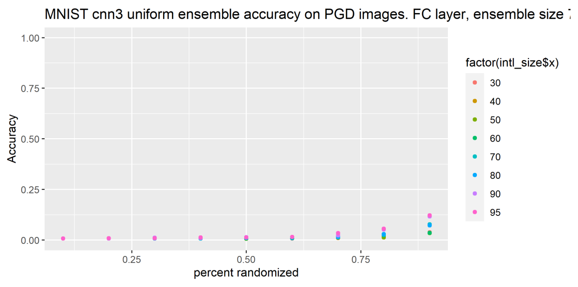
(b) Ensemble size 4



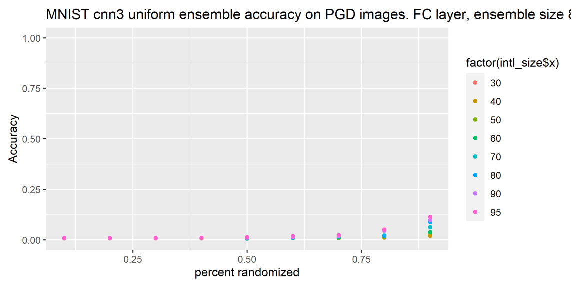
(c) Ensemble size 5



(d) Ensemble size 6

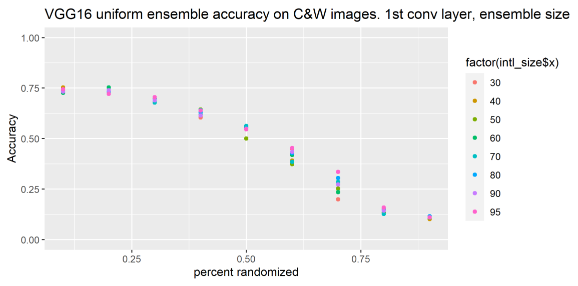


(e) Ensemble size 7

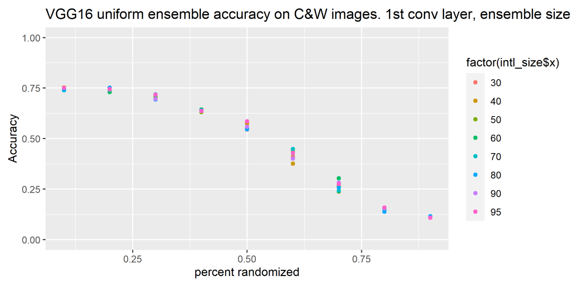


(f) Ensemble size 8

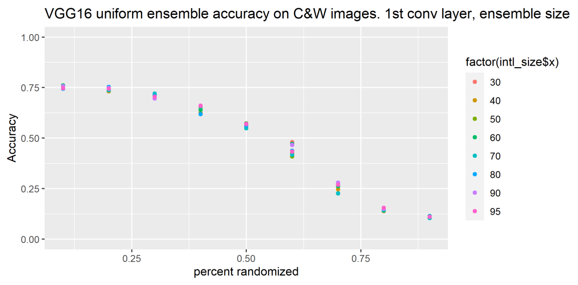
Figure 4.16. Ensemble results for Uniform randomization of CNN3, fully connected layer, on PGD attack images



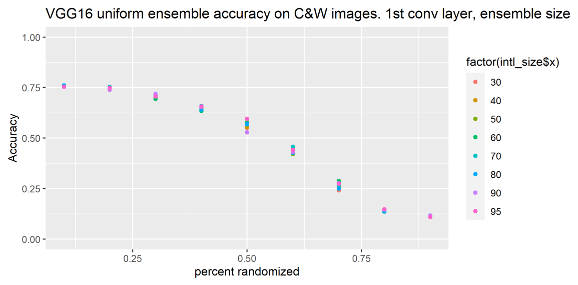
(a) Ensemble size 3



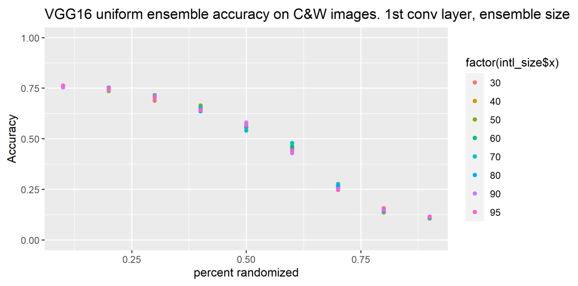
(b) Ensemble size 4



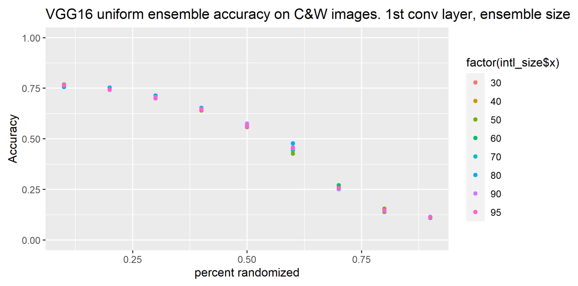
(c) Ensemble size 5



(d) Ensemble size 6



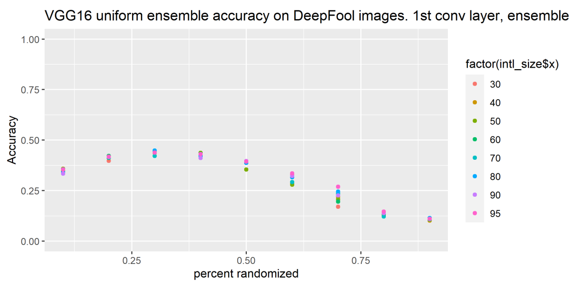
(e) Ensemble size 7



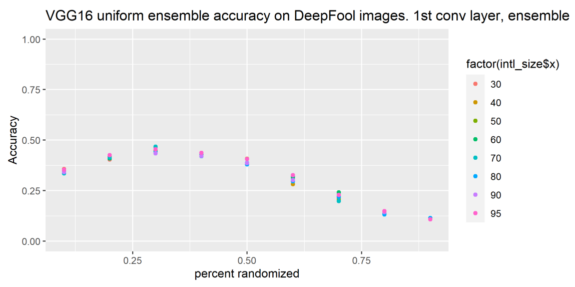
(f) Ensemble size 8

Figure 4.17. Ensemble results for Uniform randomization of VGG16, first convolutional layer, on C&W attack images

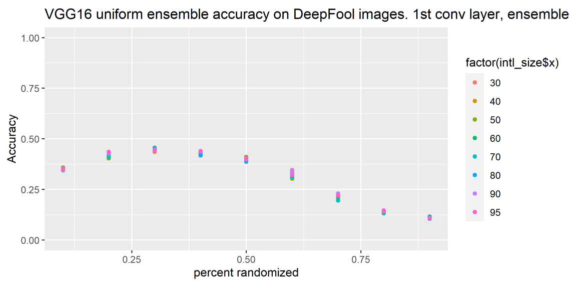
In Figure 4.17, we see accuracy on the C&W attack images with VGG16 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the first convolutional layer. Ensemble size 3 has the highest accuracy 75.3%, with 40% interval. Ensemble size 4 has the highest accuracy 75.3%, with 30% interval. Ensemble size 5 has the highest accuracy 76.0%, with 60% interval. Ensemble size 6 has the highest accuracy 76.1%, with 80% interval. Ensemble size 7 has the highest accuracy 76.3%, with 95% interval. Ensemble size 8 has the highest accuracy 76.8%, with 50% interval.



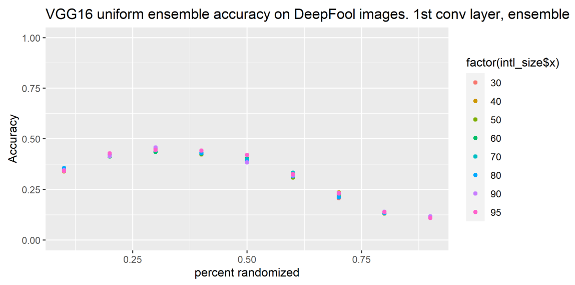
(a) Ensemble size 3



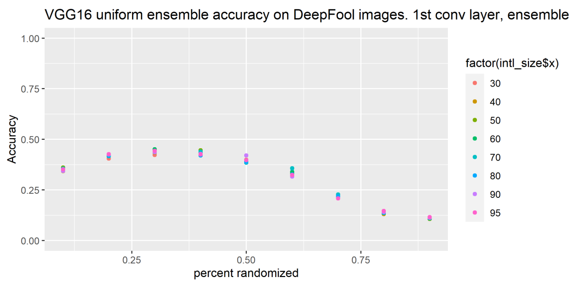
(b) Ensemble size 4



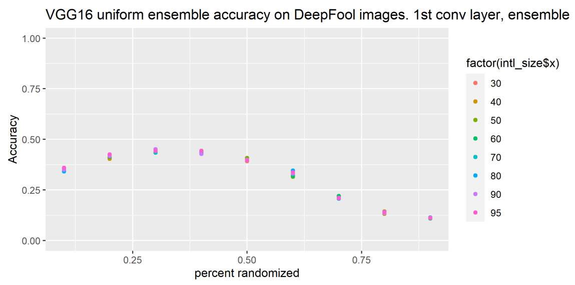
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

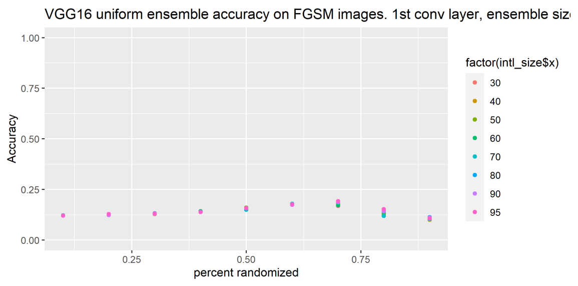


(f) Ensemble size 8

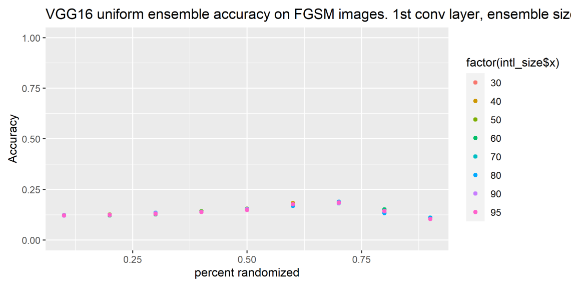
Figure 4.18. Ensemble results for Uniform randomization of VGG16, first convolutional layer, on DeepFool attack images

In Figure 4.18, we see accuracy on the DeepFool attack images with VGG16 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 30% of the weights in the first convolutional layer. Ensemble size 3 has the highest accuracy 44.8%, with 80% interval. Ensemble size 4 has the highest accuracy 46.8%, with 70% interval. Ensemble size 5 has the highest accuracy 45.7%,

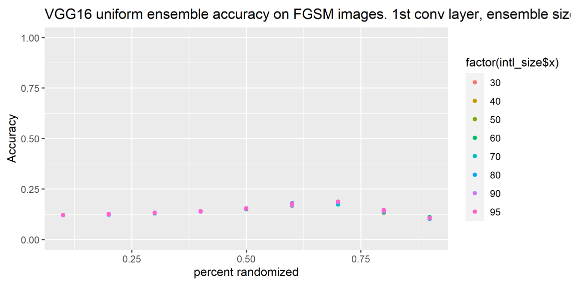
with 70% interval. Ensemble size 6 has the highest accuracy 45.7%, with 90% interval. Ensemble size 7 has the highest accuracy 45.1%, with 60% interval. Ensemble size 8 has the highest accuracy 45.1%, with 90% interval.



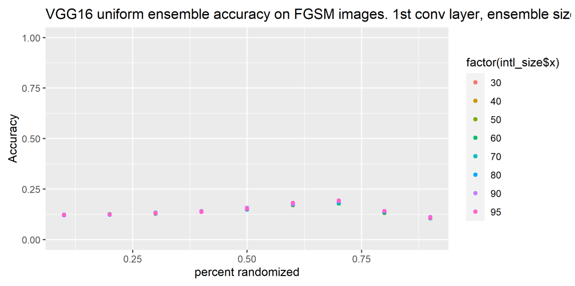
(a) Ensemble size 3



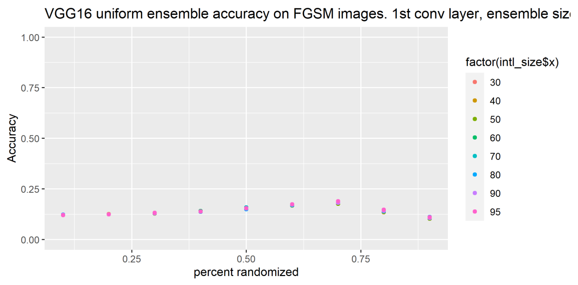
(b) Ensemble size 4



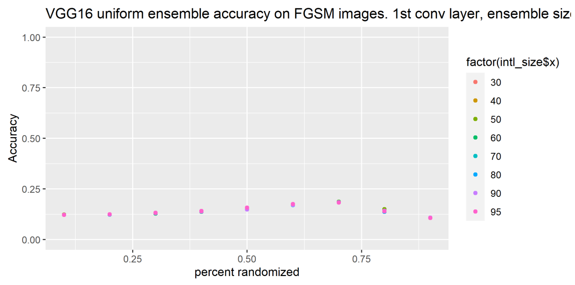
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

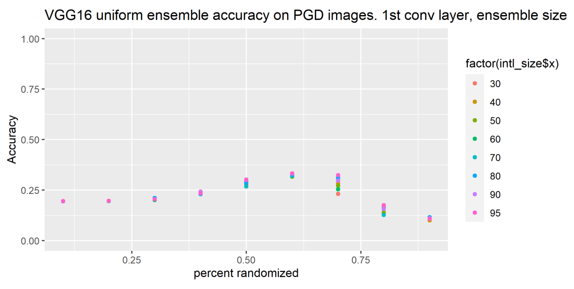


(f) Ensemble size 8

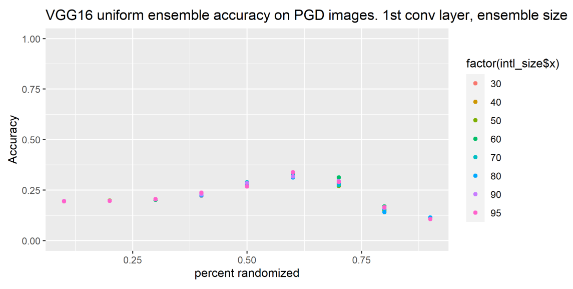
Figure 4.19. Ensemble results for Uniform randomization of VGG16, first convolutional layer, on FGSM attack images

In Figure 4.19, we see accuracy on the FGSM attack images with VGG16 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 70% of the weights in the first convolutional layer. Ensemble size 3 has the highest accuracy 19.2%, with 95% interval. Ensemble size 4 has the

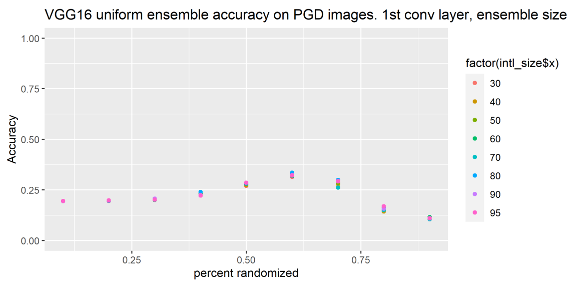
highest accuracy 18.9%, with 80% interval. Ensemble size 5 has the highest accuracy 18.8%, with 95% interval. Ensemble size 6 has the highest accuracy 19.3%, with 95% interval. Ensemble size 7 has the highest accuracy 19.0%, with 30% interval. Ensemble size 8 has the highest accuracy 18.7%, with 40% interval.



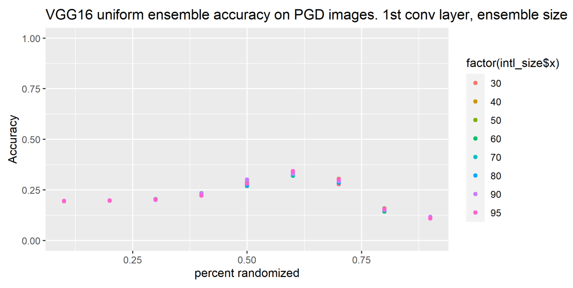
(a) Ensemble size 3



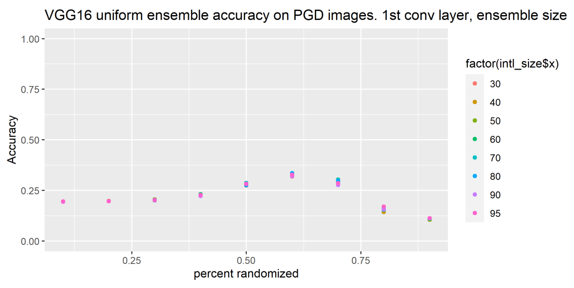
(b) Ensemble size 4



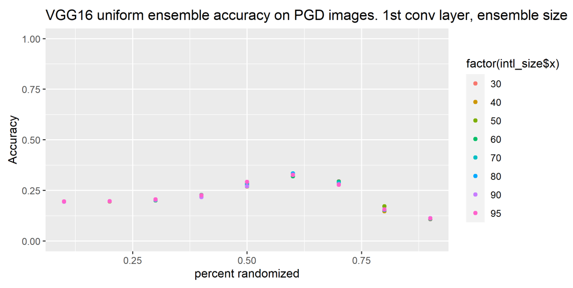
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

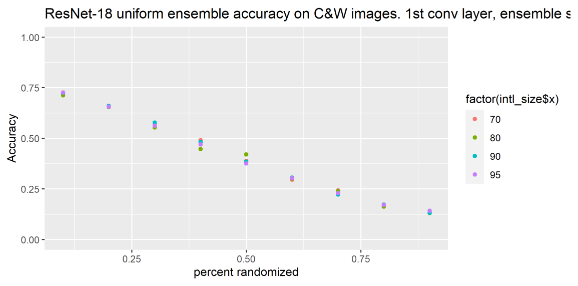


(f) Ensemble size 8

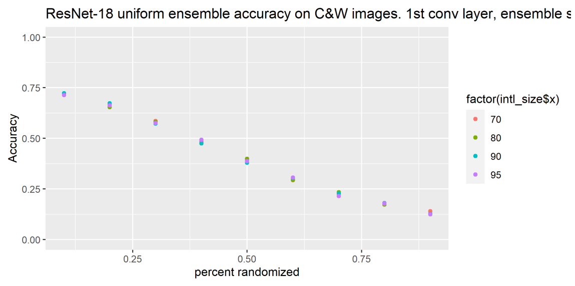
Figure 4.20. Ensemble results for Uniform randomization of VGG16, first convolutional layer, on PGD attack images

In Figure 4.20, we see accuracy on the PGD attack images with VGG16 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 60% of the weights in the first convolutional layer.

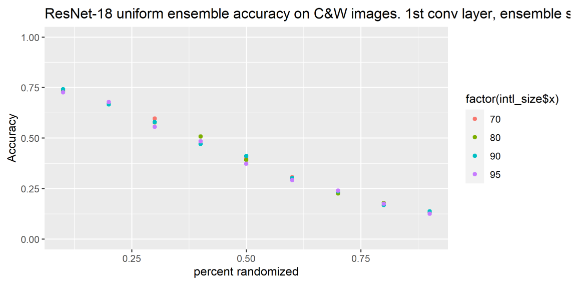
Ensemble size 3 has the highest accuracy 33.2%, with 95% interval. Ensemble size 4 has the highest accuracy 33.8%, with 95% interval. Ensemble size 5 has the highest accuracy 33.5%, with 80% interval. Ensemble size 6 has the highest accuracy 34.3%, with 95% interval. Ensemble size 7 has the highest accuracy 33.5%, with 80% interval. Ensemble size 8 has the highest accuracy 33.4%, with 70% interval.



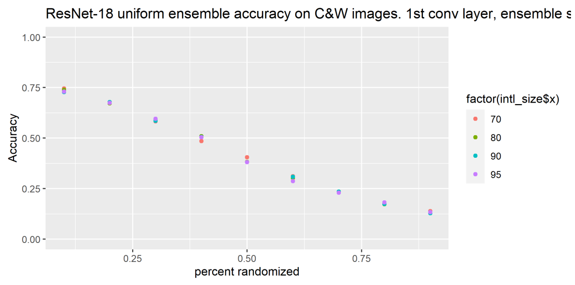
(a) Ensemble size 3



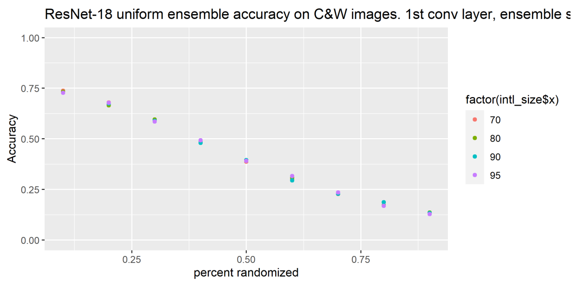
(b) Ensemble size 4



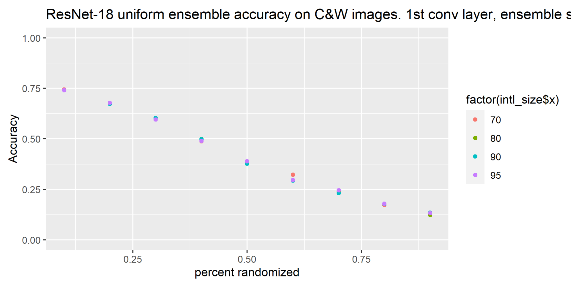
(c) Ensemble size 5



(d) Ensemble size 6



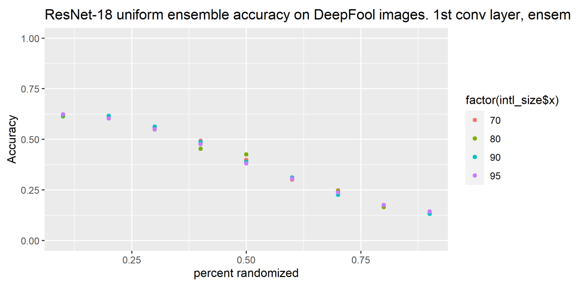
(e) Ensemble size 7



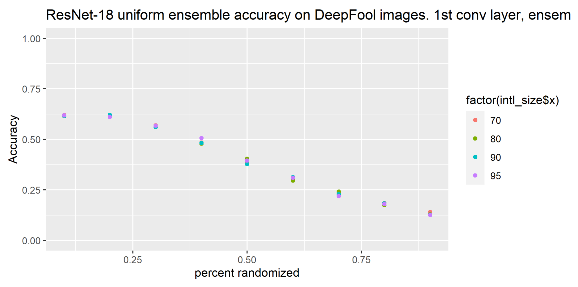
(f) Ensemble size 8

Figure 4.21. Ensemble results for Uniform randomization of ResNet18, first convolutional layer, on C&W attack images

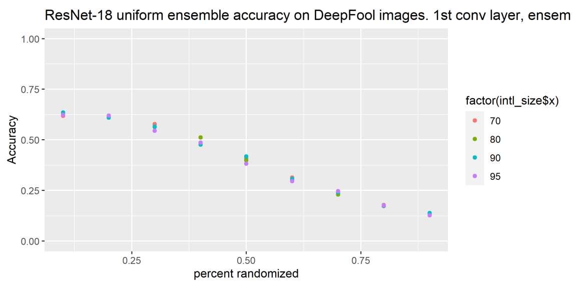
In Figure 4.21, we see accuracy on the C&W attack images with ResNet-18 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the first convolutional layer. Ensemble size 3 has the highest accuracy 72.6%, with 95% interval. Ensemble size 4 has the highest accuracy 72.3%, with 90% interval. Ensemble size 5 has the highest accuracy 74.2%, with 90% interval. Ensemble size 6 has the highest accuracy 74.6%, with 70% interval. Ensemble size 7 has the highest accuracy 73.7%, with 70% interval. Ensemble size 8 has the highest accuracy 74.4%, with 70% interval.



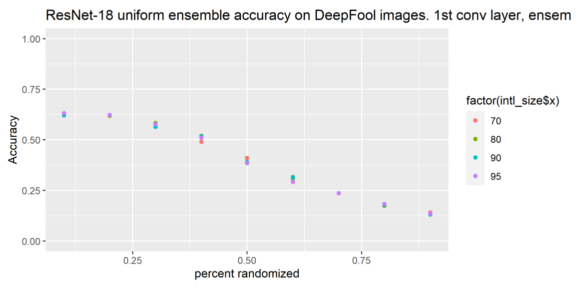
(a) Ensemble size 3



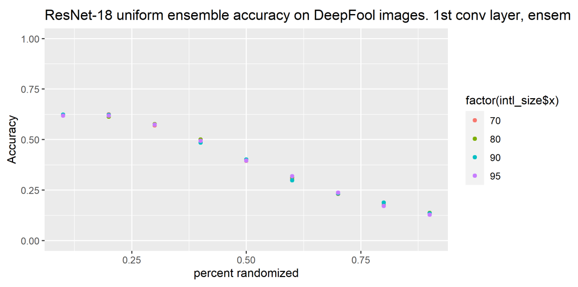
(b) Ensemble size 4



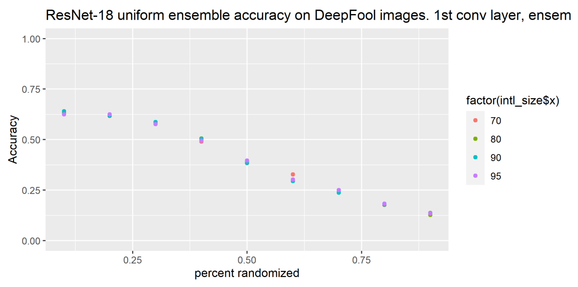
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7



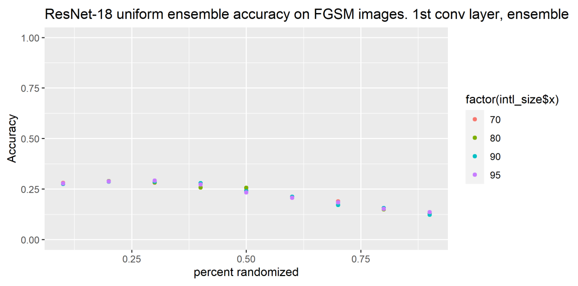
(f) Ensemble size 8

Figure 4.22. Ensemble results for Uniform randomization of ResNet18, first convolutional layer, on DeepFool attack images

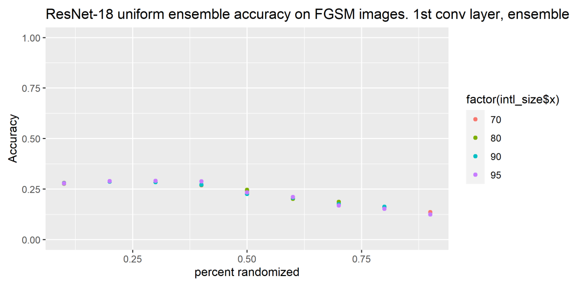
In Figure 4.22, we see accuracy on the DeepFool attack images with ResNet-18 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 62.4%, with 10% of weights in the fully connected layer randomized and 95% interval. Ensemble size 4 has the highest accuracy 62.1%, with 20% of weights in the fully connected layer randomized and 90% interval. Ensemble size 5 has the highest accuracy 63.5%, with 10% of weights in the fully connected layer randomized and 90% interval. Ensemble size 6 has the highest accuracy 63.1%, with 10% of weights in the fully connected layer randomized and 95% interval. Ensemble size 7 has the highest accuracy 62.5%, with 20% of weights in the fully connected layer randomized and 70% interval. Ensemble size 8 has the highest accuracy 64.0%, with 10% of weights in the fully connected layer randomized and 90% interval.

In Figure 4.23, we see accuracy on the FGSM attack images with ResNet-18 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 29.2%, with 30% of weights in the fully connected layer randomized and 95% interval. Ensemble size 4 has the highest accuracy 29.2%, with 30% of weights in the fully connected layer randomized and 95% interval. Ensemble size 5 has the highest accuracy 29.3%, with 20% of weights in the fully connected layer randomized and 70% interval. Ensemble size 6 has the highest accuracy 29.6%, with 20% of weights in the fully connected layer randomized and 80% interval. Ensemble size 7 has the highest accuracy 29.7%, with 30% of weights in the fully connected layer randomized and 95% interval. Ensemble size 8 has the highest accuracy 29.4%, with 30% of weights in the fully connected layer randomized and 95% interval.

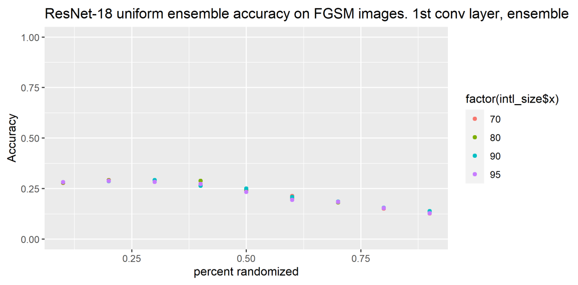
In Figure 4.24, we see accuracy on the PGD attack images with ResNet-18 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 35.1%, with 30% of weights in the fully connected layer randomized and 90% interval. Ensemble size 4 has the highest accuracy 35.7%, with 40% of weights in the fully connected layer randomized and 95% interval. Ensemble size 5 has the highest accuracy 35.0%, with 30% of weights in the fully connected layer randomized and 70% interval. Ensemble size 6 has the highest accuracy 35.9%, with 40% of weights in the fully connected layer randomized and 80% interval. Ensemble size 7 has the highest accuracy 35.6%, with



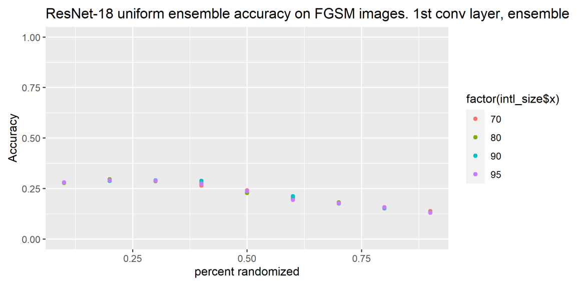
(a) Ensemble size 3



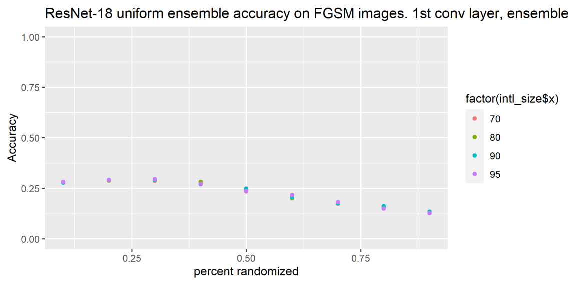
(b) Ensemble size 4



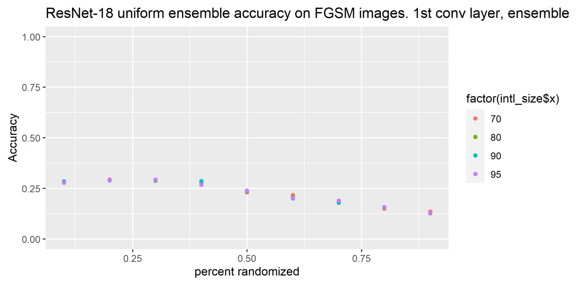
(c) Ensemble size 5



(d) Ensemble size 6



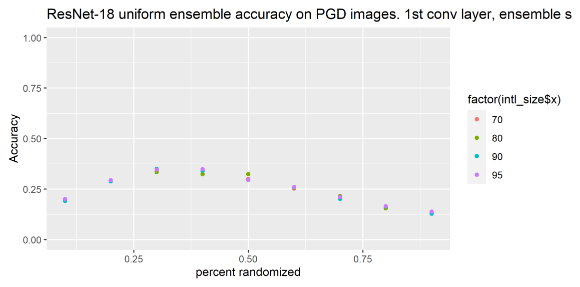
(e) Ensemble size 7



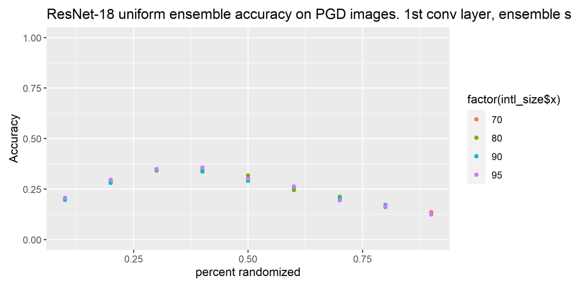
(f) Ensemble size 8

Figure 4.23. Ensemble results for Uniform randomization of ResNet18, first convolutional layer, on FGSM attack images

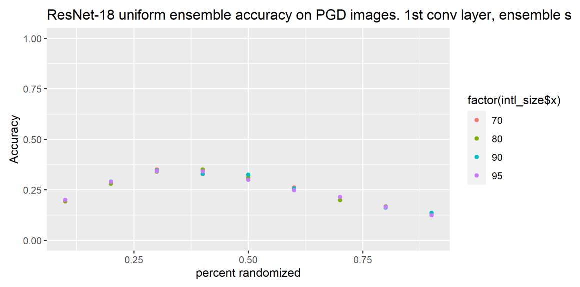
30% of weights in the fully connected layer randomized and 95% interval. Ensemble size 8 has the highest accuracy 35.7%, with 40% of weights in the fully connected layer randomized and 90% interval.



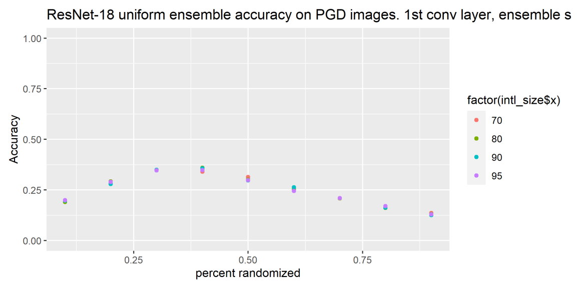
(a) Ensemble size 3



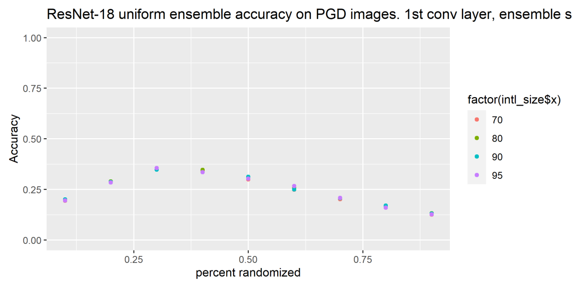
(b) Ensemble size 4



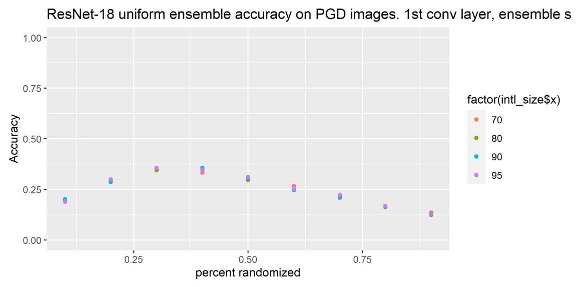
(c) Ensemble size 5



(d) Ensemble size 6

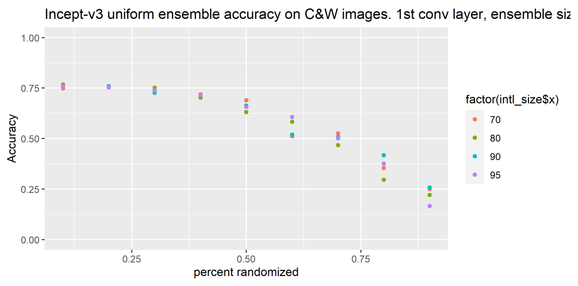


(e) Ensemble size 7

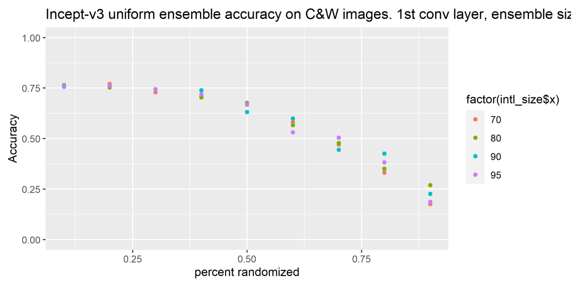


(f) Ensemble size 8

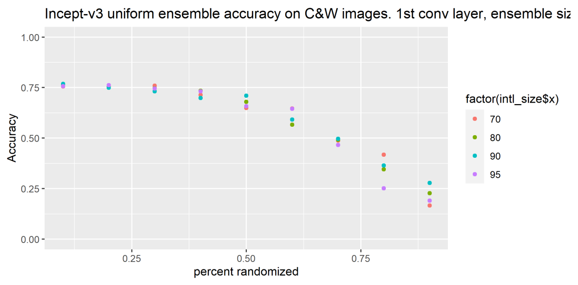
Figure 4.24. Ensemble results for Uniform randomization of ResNet18, first convolutional layer, on PGD attack images



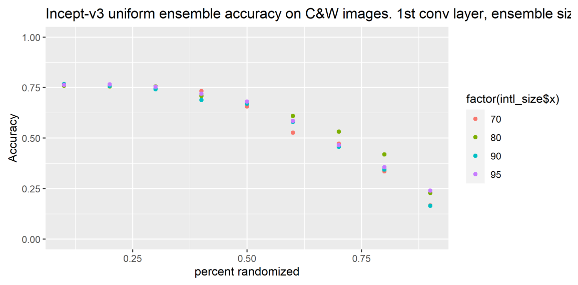
(a) Ensemble size 3



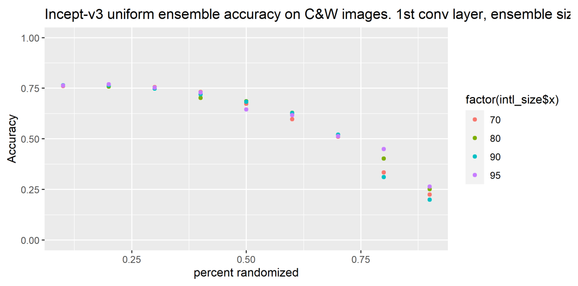
(b) Ensemble size 4



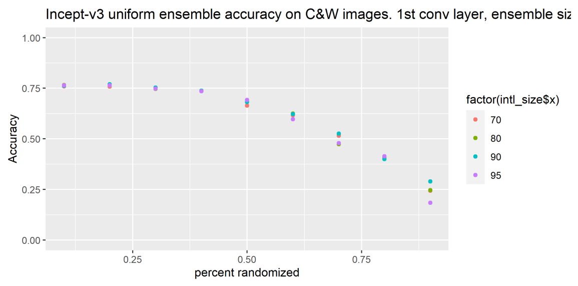
(c) Ensemble size 5



(d) Ensemble size 6



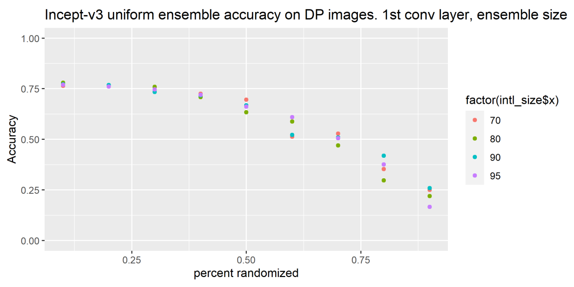
(e) Ensemble size 7



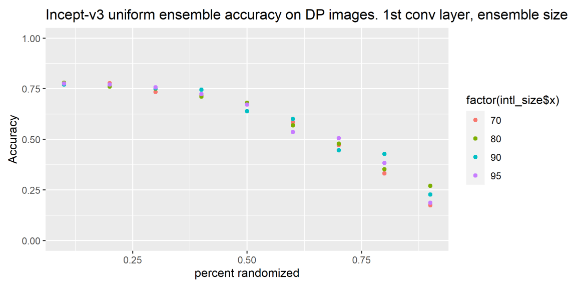
(f) Ensemble size 8

Figure 4.25. Ensemble results for Uniform randomization of Inception v3, first convolutional layer, on C&W attack images

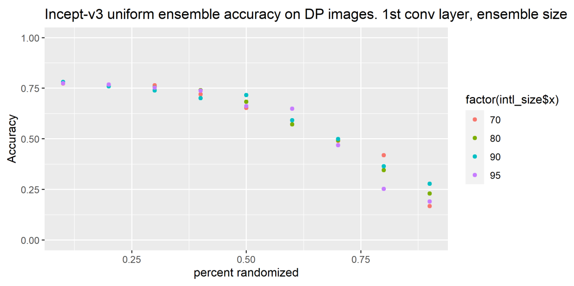
In Figure 4.21, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 76.7%, with 10% of weights in the fully connected layer randomized and 80% interval. Ensemble size 4 has the highest accuracy 77.0%, with 20% of weights in the fully connected layer randomized and 70% interval. Ensemble size 5 has the highest accuracy 76.9%, with 10% of weights in the fully connected layer randomized and 90% interval. Ensemble size 6 has the highest accuracy 76.7%, with 10% of weights in the fully connected layer randomized and 90% interval. Ensemble size 7 has the highest accuracy 76.9%, with 20% of weights in the fully connected layer randomized and 95% interval. Ensemble size 8 has the highest accuracy 76.9%, with 20% of weights in the fully connected layer randomized and 90% interval.



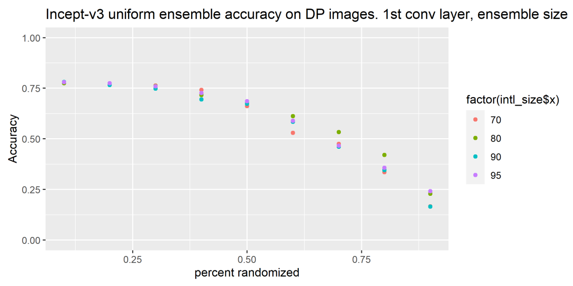
(a) Ensemble size 3



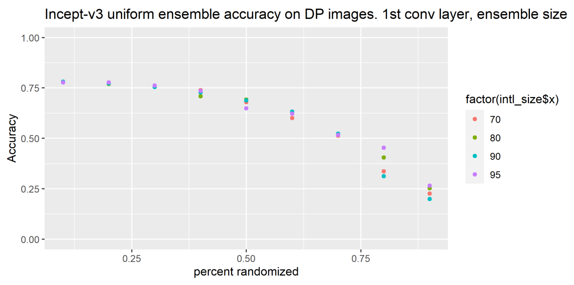
(b) Ensemble size 4



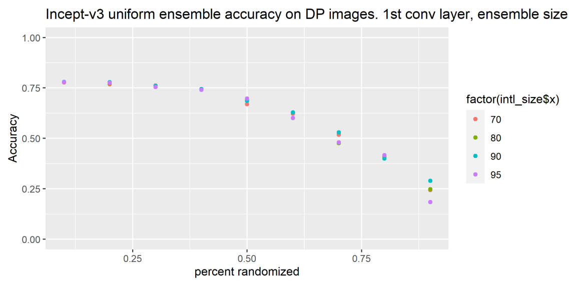
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

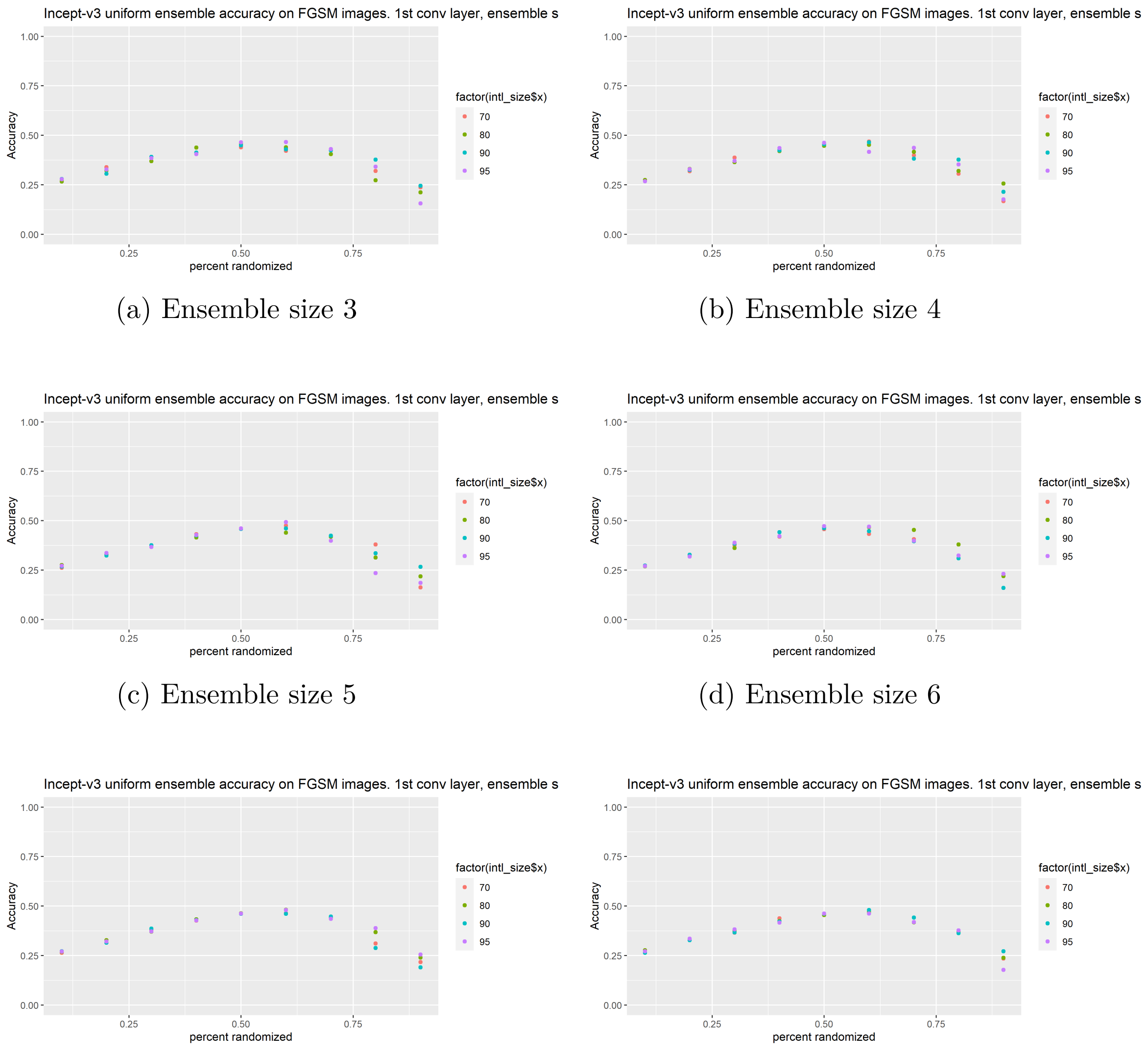


(f) Ensemble size 8

Figure 4.26. Ensemble results for Uniform randomization of Inception v3, first convolutional layer, on DeepFool attack images

In Figure 4.26, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the first convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the first convolutional layer.

Ensemble size 3 has the highest accuracy 78.0%, with 80% interval. Ensemble size 4 has the highest accuracy 78.0%, with 80% interval. Ensemble size 5 has the highest accuracy 78.2%, with 90% interval. Ensemble size 6 has the highest accuracy 78.1%, with 90% interval. Ensemble size 7 has the highest accuracy 78.1%, with 90% interval. Ensemble size 8 has the highest accuracy 78.0%, with 90% interval.



(a) Ensemble size 3 (b) Ensemble size 4

(c) Ensemble size 5 (d) Ensemble size 6

(e) Ensemble size 7 (f) Ensemble size 8

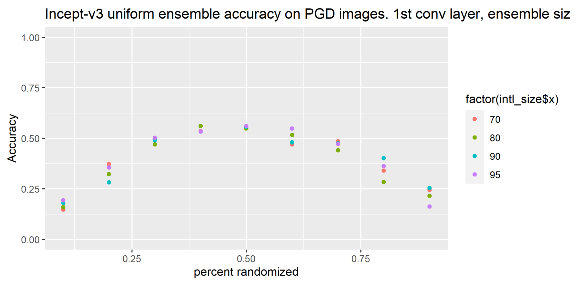
Figure 4.27. Ensemble results for Uniform randomization of Inception v3, first convolutional layer, on FGSM attack images

In Figure 4.23, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 46.6%, with 60% of weights in the fully connected layer randomized and 95% interval. Ensemble size 4 has the highest accuracy 46.9%, with 60% of weights in the fully connected layer randomized and 70% interval. Ensemble size 5 has the highest accuracy 49.3%, with 60% of weights in the fully connected layer randomized and 95% interval. Ensemble size 6 has the highest accuracy 47.2%, with 50% of weights in the fully connected layer randomized and 95% interval. Ensemble size 7 has the highest accuracy 48.2%, with 60% of weights in the fully connected layer randomized and 80% interval. Ensemble size 8 has the highest accuracy 48.0%, with 60% of weights in the fully connected layer randomized and 90% interval.

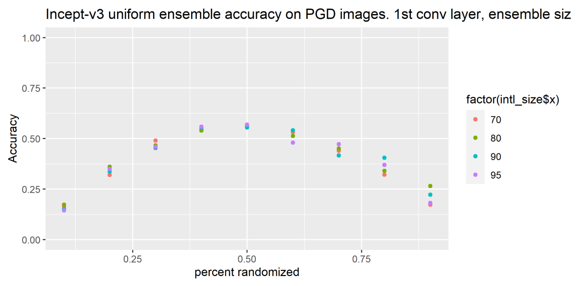
In Figure 4.24, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the first convolutional layer. Ensemble size 3 has the highest accuracy 56.2%, with 40% of weights in the fully connected layer randomized and 80% interval. Ensemble size 4 has the highest accuracy 57.0%, with 50% of weights in the fully connected layer randomized and 95% interval. Ensemble size 5 has the highest accuracy 58.4%, with 50% of weights in the fully connected layer randomized and 90% interval. Ensemble size 6 has the highest accuracy 58.5%, with 50% of weights in the fully connected layer randomized and 95% interval. Ensemble size 7 has the highest accuracy 58.1%, with 50% of weights in the fully connected layer randomized and 80% interval. Ensemble size 8 has the highest accuracy 58.1%, with 50% of weights in the fully connected layer randomized and 95% interval.

In Figure 4.29, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the second convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the second convolutional layer. Ensemble size 3 has the highest accuracy 58.2%, with 70% interval. Ensemble size 4 has the highest accuracy 46.5%, with 95% interval. Ensemble size 5 has the highest accuracy 51.9%, with 90% interval. Ensemble size 6 has the highest accuracy 51.9%, with 95% interval. Ensemble size 7 has the highest accuracy 47.9%, with 80% interval. Ensemble size 8 has the highest accuracy 51.0%, with 80% interval.

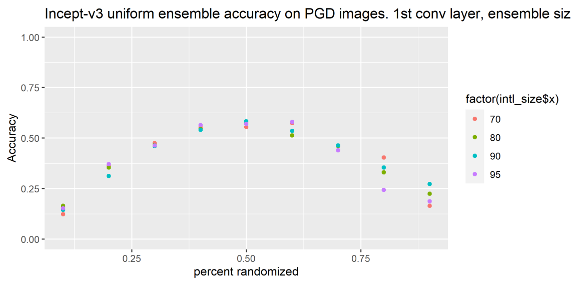
In Figure 4.30, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the second convolutional layer. For all ensemble sizes, the highest



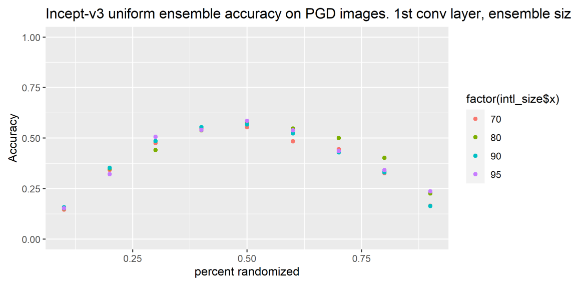
(a) Ensemble size 3



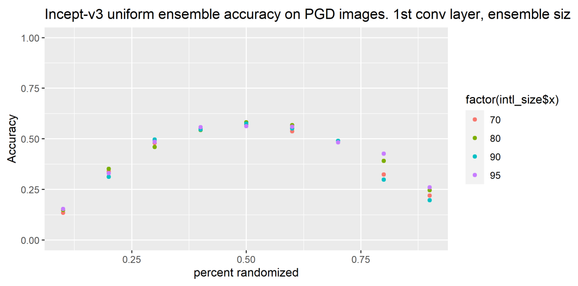
(b) Ensemble size 4



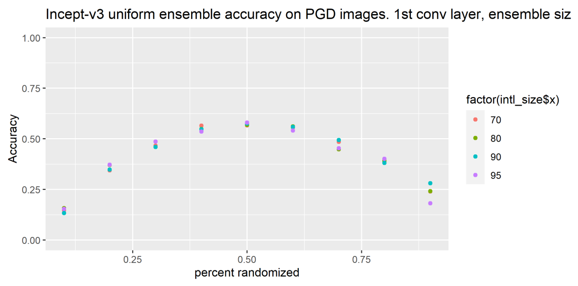
(c) Ensemble size 5



(d) Ensemble size 6



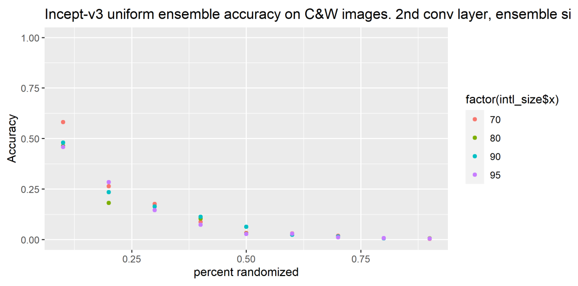
(e) Ensemble size 7



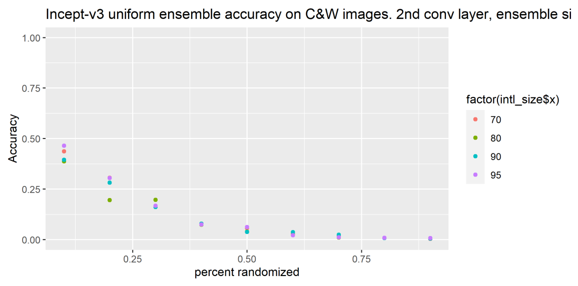
(f) Ensemble size 8

Figure 4.28. Ensemble results for Uniform randomization of Inception v3, first convolutional layer, on PGD attack images

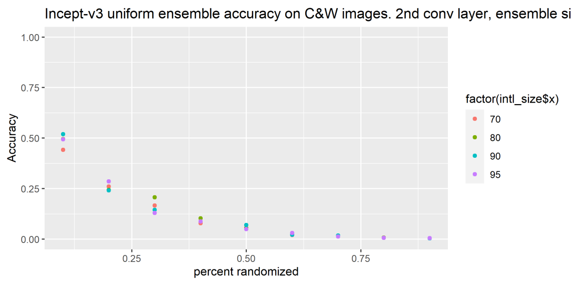
accuracy is achieved when randomizing 10% of the weights in the second convolutional layer. Ensemble size 3 has the highest accuracy 58.7%, with 70% interval. Ensemble size 4 has the highest accuracy 46.7%, with 95% interval. Ensemble size 5 has the highest accuracy 52.2%, with 90% interval. Ensemble size 6 has the highest accuracy 52.2%, with 95% interval. Ensemble size 7 has the highest accuracy 48.2%, with 80% interval. Ensemble size 8 has the highest accuracy 51.4%, with 80% interval.



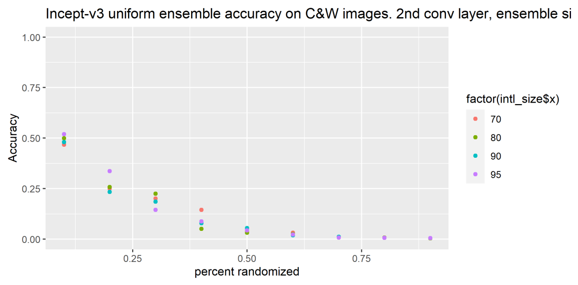
(a) Ensemble size 3



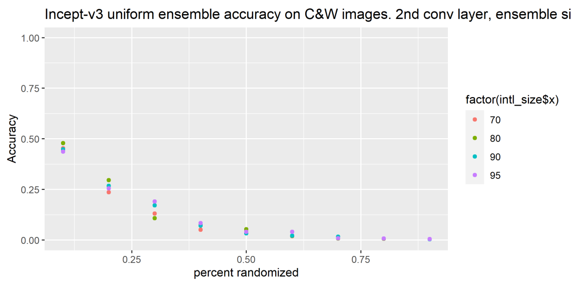
(b) Ensemble size 4



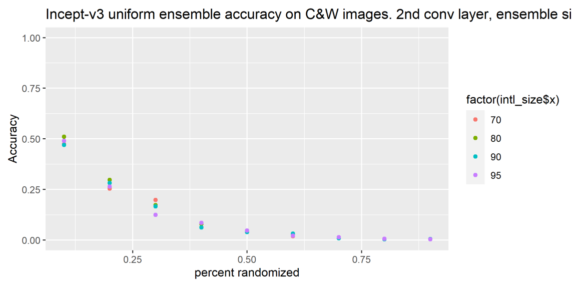
(c) Ensemble size 5



(d) Ensemble size 6



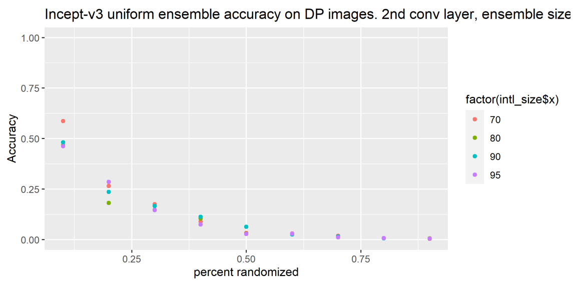
(e) Ensemble size 7



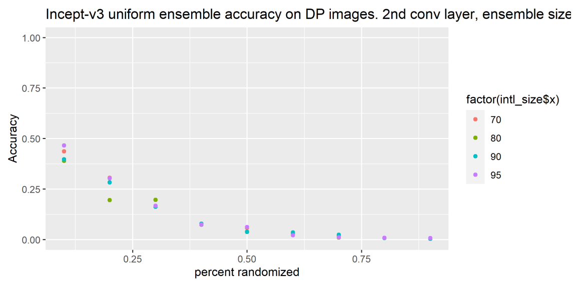
(f) Ensemble size 8

Figure 4.29. Ensemble results for Uniform randomization of Inception v3, second convolutional layer, on C&W attack images

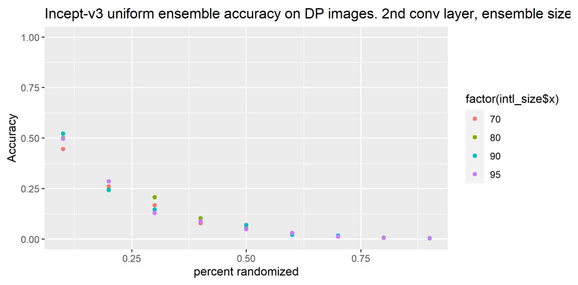
In Figure 4.31, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the second convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the second convolutional layer. Ensemble size 3 has the highest accuracy 43.9%, with 70% interval. Ensemble size 4 has the highest accuracy 38.4%, with 95% interval. Ensemble size 5 has the highest accuracy 42.2%, with 90% interval. Ensemble size 6 has the highest accuracy 42.5%, with 95% interval.



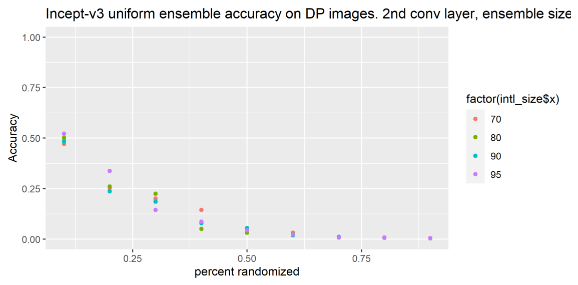
(a) Ensemble size 3



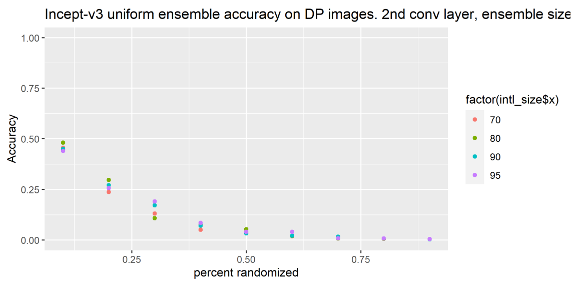
(b) Ensemble size 4



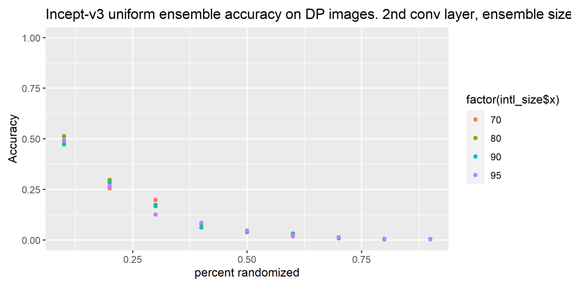
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

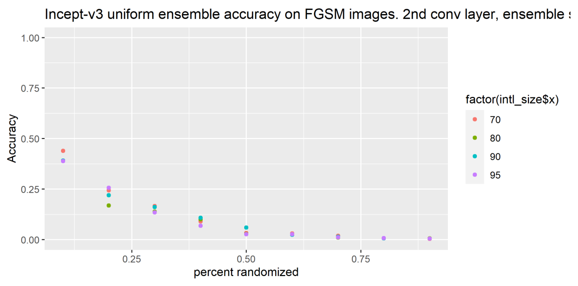


(f) Ensemble size 8

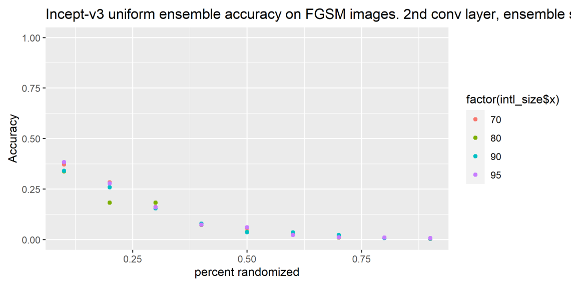
Figure 4.30. Ensemble results for Uniform randomization of Inception v3, second convolutional layer, on DeepFool attack images

Ensemble size 7 has the highest accuracy 40.3%, with 80% interval. Ensemble size 8 has the highest accuracy 42.6%, with 80% interval.

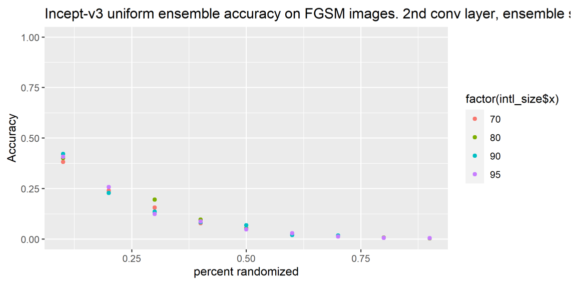
In Figure 4.32, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the second convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the second convolutional layer. Ensemble size 3 has the highest accuracy 51.8%, with 70% interval. Ensemble size 4 has the highest accuracy 43.5%, with 95% interval. Ensemble size 5 has the highest accuracy 47.3%,



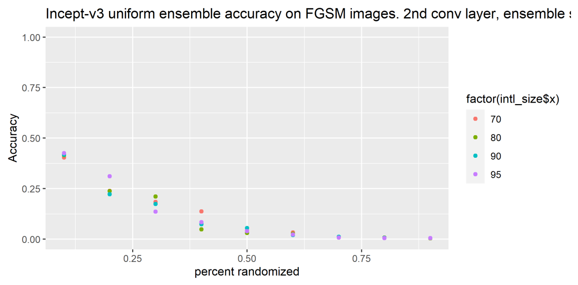
(a) Ensemble size 3



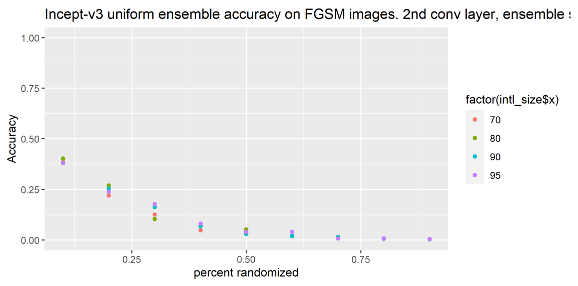
(b) Ensemble size 4



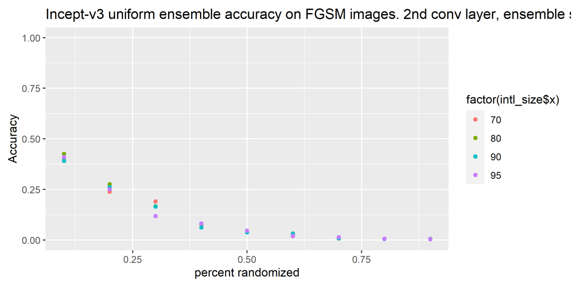
(c) Ensemble size 5



(d) Ensemble size 6



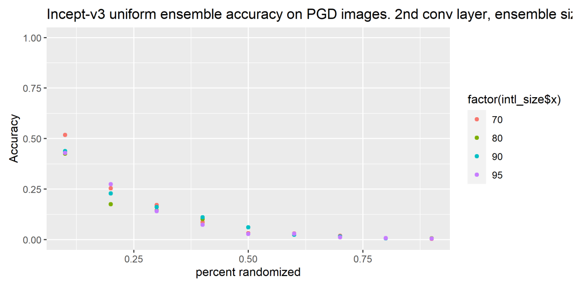
(e) Ensemble size 7



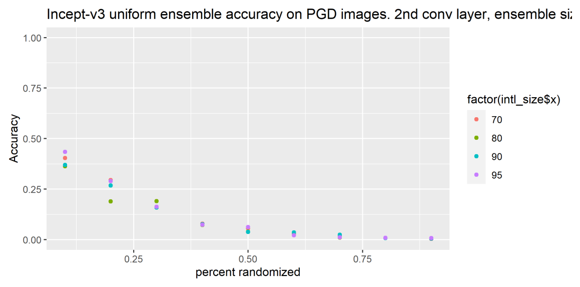
(f) Ensemble size 8

Figure 4.31. Ensemble results for Uniform randomization of Inception v3, second convolutional layer, on FGSM attack images

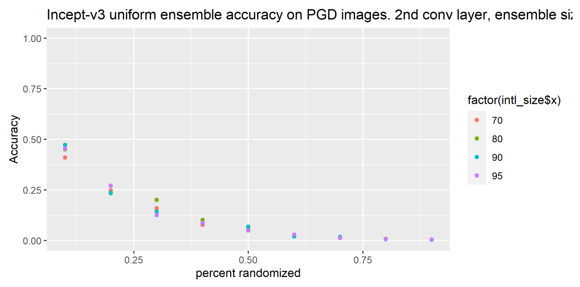
with 90% interval. Ensemble size 6 has the highest accuracy 47.3%, with 95% interval. Ensemble size 7 has the highest accuracy 44.3%, with 80% interval. Ensemble size 8 has the highest accuracy 47.3%, with 80% interval.



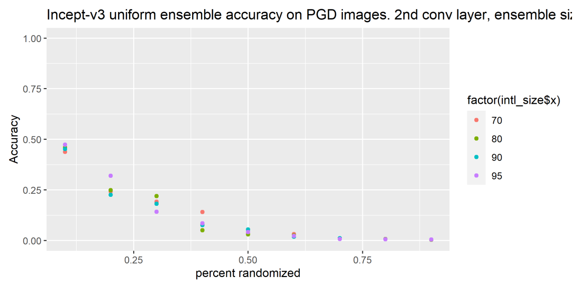
(a) Ensemble size 3



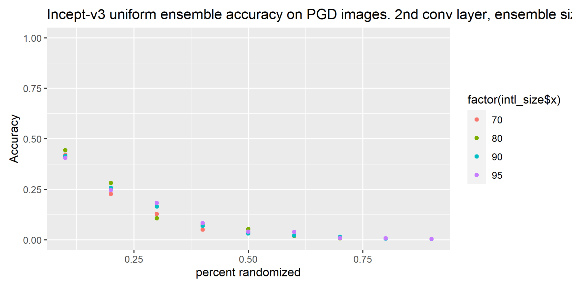
(b) Ensemble size 4



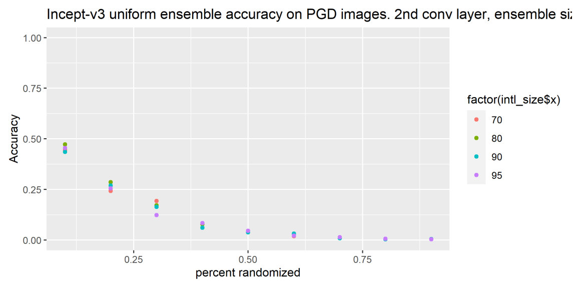
(c) Ensemble size 5



(d) Ensemble size 6

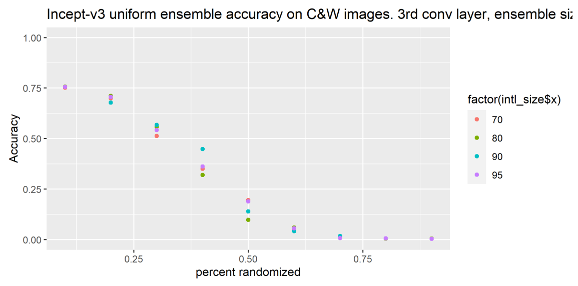


(e) Ensemble size 7

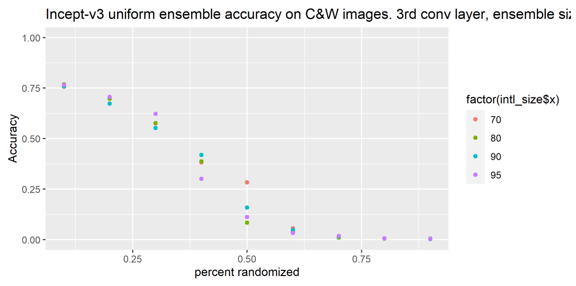


(f) Ensemble size 8

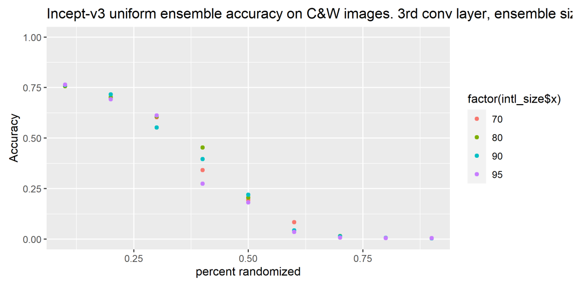
Figure 4.32. Ensemble results for Uniform randomization of Inception v3, second convolutional layer, on PGD attack images



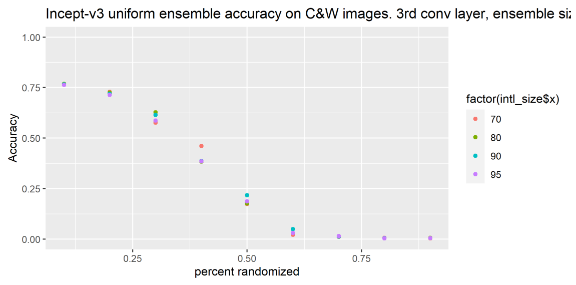
(a) Ensemble size 3



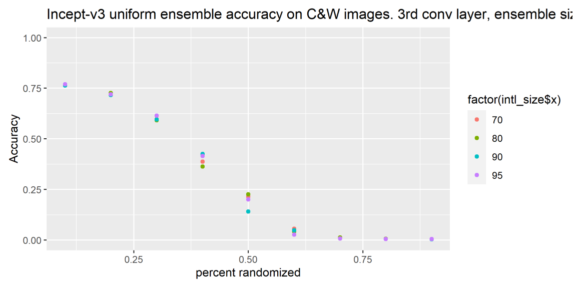
(b) Ensemble size 4



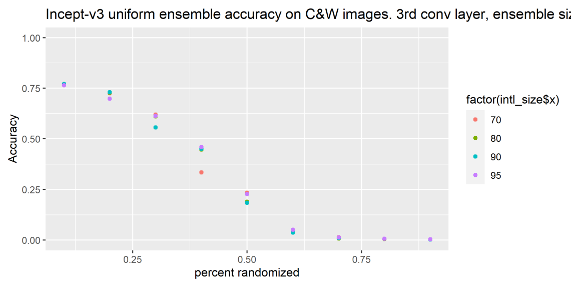
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

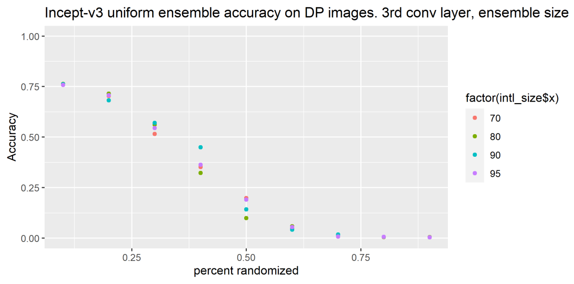


(f) Ensemble size 8

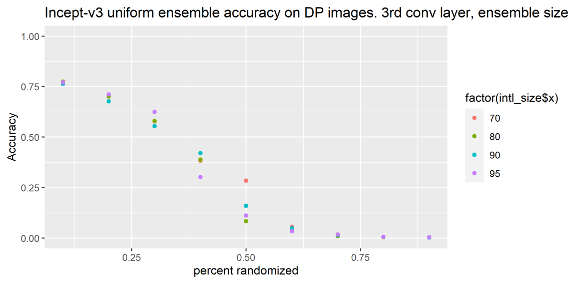
Figure 4.33. Ensemble results for Uniform randomization of Inception v3, third convolutional layer, on C&W attack images

In Figure 4.33, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the third convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the third convolutional layer. Ensemble size 3 has the highest accuracy 75.7%, with 80% interval. Ensemble size 4 has the highest accuracy 76.9%, with 80% interval. Ensemble size 5 has the highest accuracy 76.5%, with 70% interval. Ensemble size 6 has the highest accuracy 76.9%, with 80% interval.

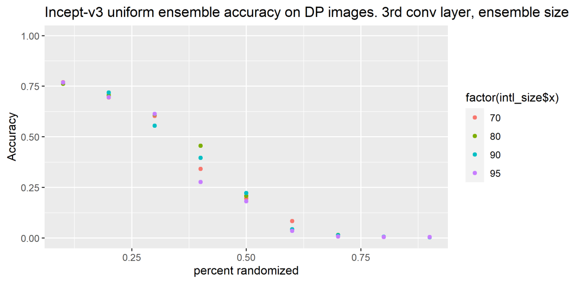
Ensemble size 7 has the highest accuracy 77.0%, with 95% interval. Ensemble size 8 has the highest accuracy 77.1%, with 90% interval.



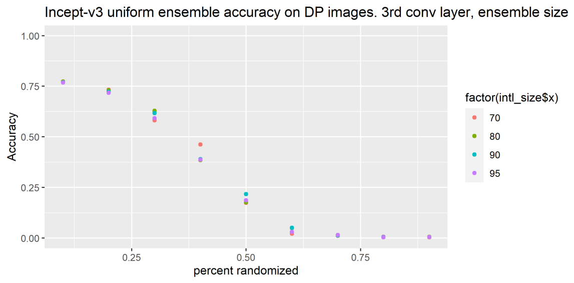
(a) Ensemble size 3



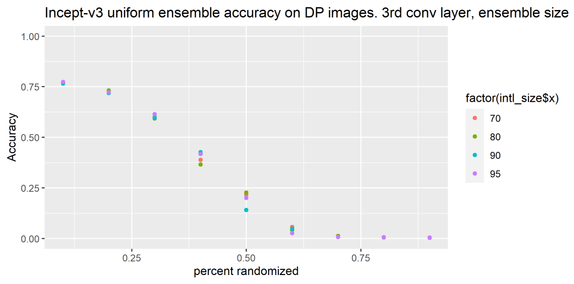
(b) Ensemble size 4



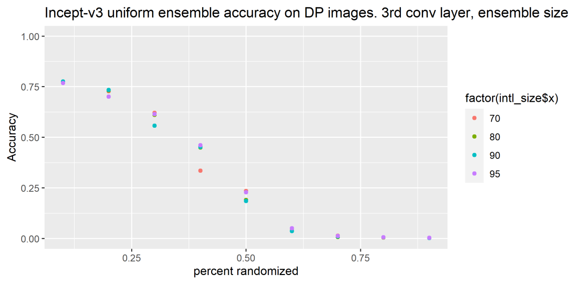
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

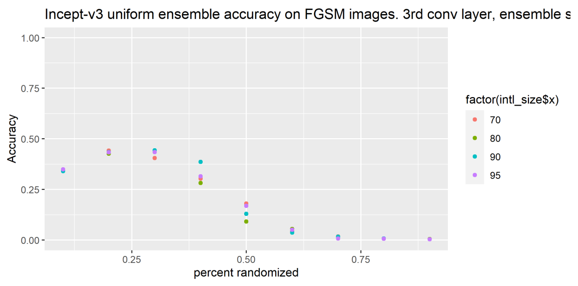


(f) Ensemble size 8

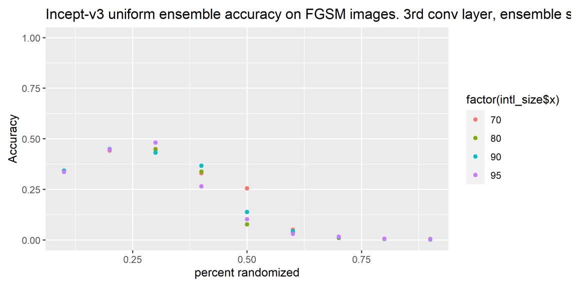
Figure 4.34. Ensemble results for Uniform randomization of Inception v3, third convolutional layer, on DeepFool attack images

In Figure 4.34, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the third convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the third convolutional layer. Ensemble size 3 has the highest accuracy 76.3%, with 80% interval. Ensemble size 4 has the highest accuracy 77.4%, with 80% interval. Ensemble size 5 has the highest accuracy 76.9%,

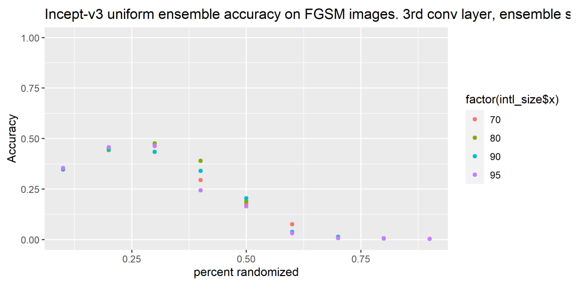
with 70% interval. Ensemble size 6 has the highest accuracy 77.3%, with 80% interval. Ensemble size 7 has the highest accuracy 77.3%, with 95% interval. Ensemble size 8 has the highest accuracy 77.5%, with 90% interval.



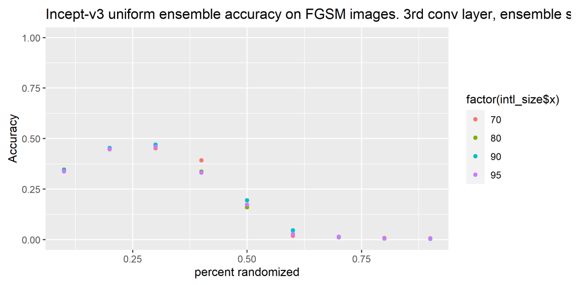
(a) Ensemble size 3



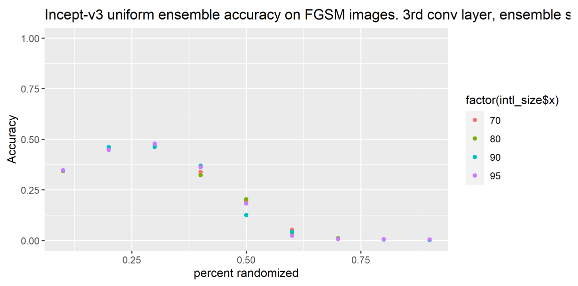
(b) Ensemble size 4



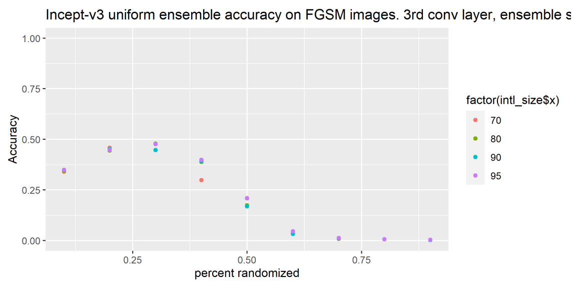
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

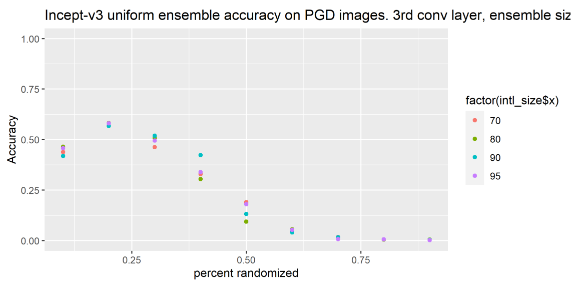


(f) Ensemble size 8

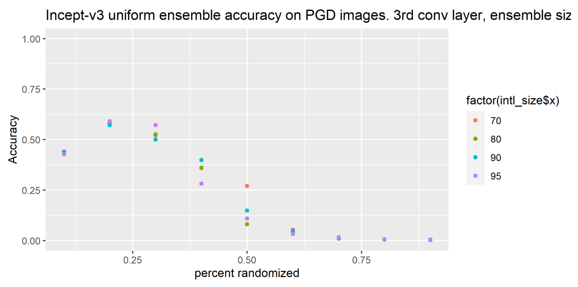
Figure 4.35. Ensemble results for Uniform randomization of Inception v3, third convolutional layer, on FGSM attack images

In Figure 4.35, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the third convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 30% of the weights in the third convolutional layer. Ensemble size 3 has the highest accuracy 44.3%, with 90% interval. Ensemble size 4 has the

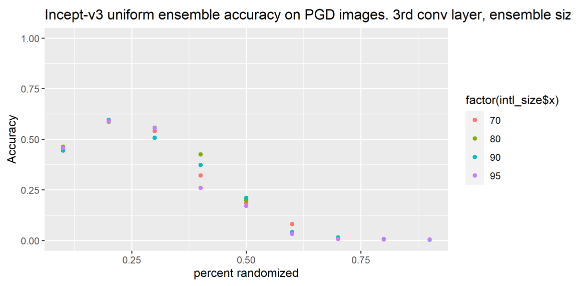
highest accuracy 48.1%, with 95% interval. Ensemble size 5 has the highest accuracy 47.6%, with 80% interval. Ensemble size 6 has the highest accuracy 47.1%, with 90% interval. Ensemble size 7 has the highest accuracy 47.8%, with 95% interval. Ensemble size 8 has the highest accuracy 48.0%, with 80% interval.



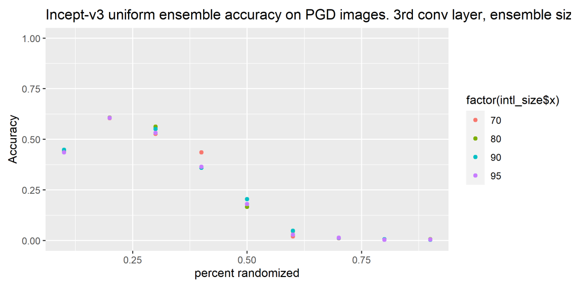
(a) Ensemble size 3



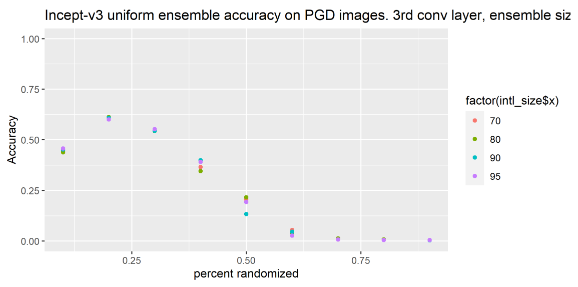
(b) Ensemble size 4



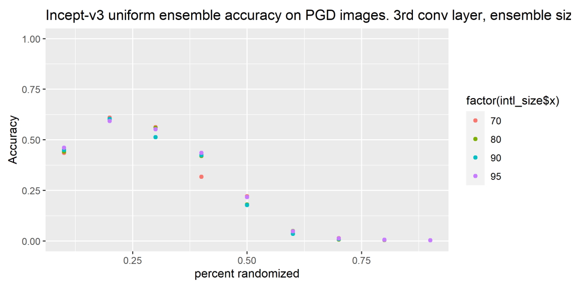
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

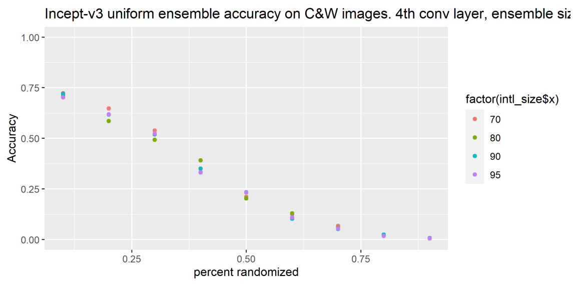


(f) Ensemble size 8

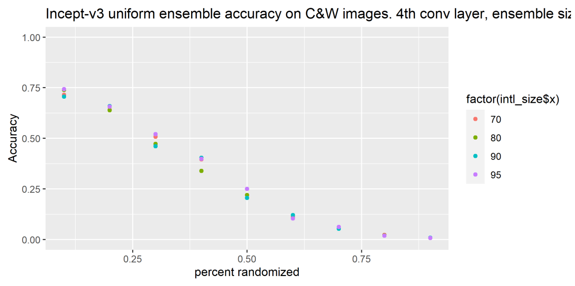
Figure 4.36. Ensemble results for Uniform randomization of Inception v3, third convolutional layer, on PGD attack images

In Figure 4.36, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the third convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 20% of the weights in the third convolutional layer.

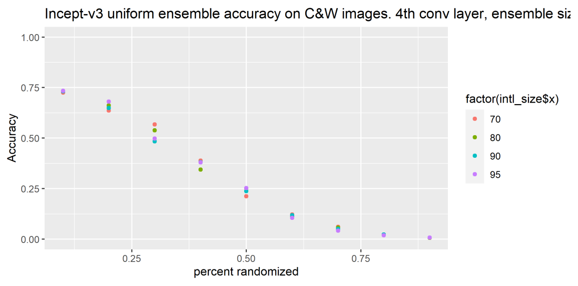
Ensemble size 3 has the highest accuracy 58.1%, with 80% interval. Ensemble size 4 has the highest accuracy 59.1%, with 95% interval. Ensemble size 5 has the highest accuracy 59.5%, with 90% interval. Ensemble size 6 has the highest accuracy 60.7%, with 80% interval. Ensemble size 7 has the highest accuracy 61.2%, with 80% interval. Ensemble size 8 has the highest accuracy 60.9%, with 70% interval.



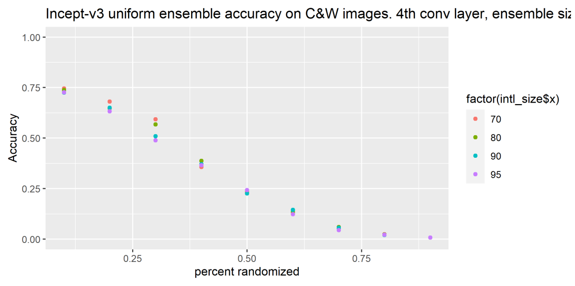
(a) Ensemble size 3



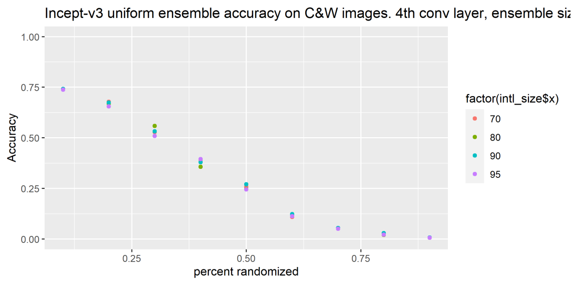
(b) Ensemble size 4



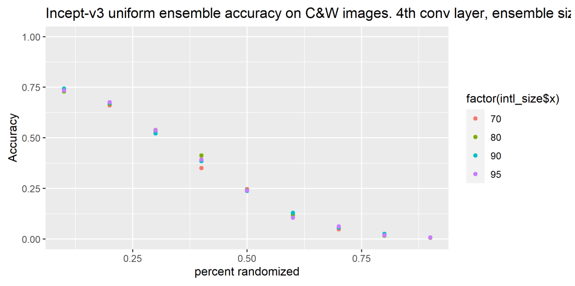
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7



(f) Ensemble size 8

Figure 4.37. Ensemble results for Uniform randomization of Inception v3, fourth convolutional layer, on C&W attack images

In Figure 4.37, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the fourth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the fourth convolutional layer. Ensemble size 3 has the highest accuracy 72.3%, with 70% interval. Ensemble size 4 has the highest accuracy 74.3%, with 95% interval. Ensemble size 5 has the highest accuracy 73.4%, with 95% interval. Ensemble size 6 has the highest accuracy 74.5%, with 70% interval. Ensemble size 7 has the highest accuracy 74.1%, with 80% interval. Ensemble size 8 has the highest accuracy 74.3%, with 90% interval.

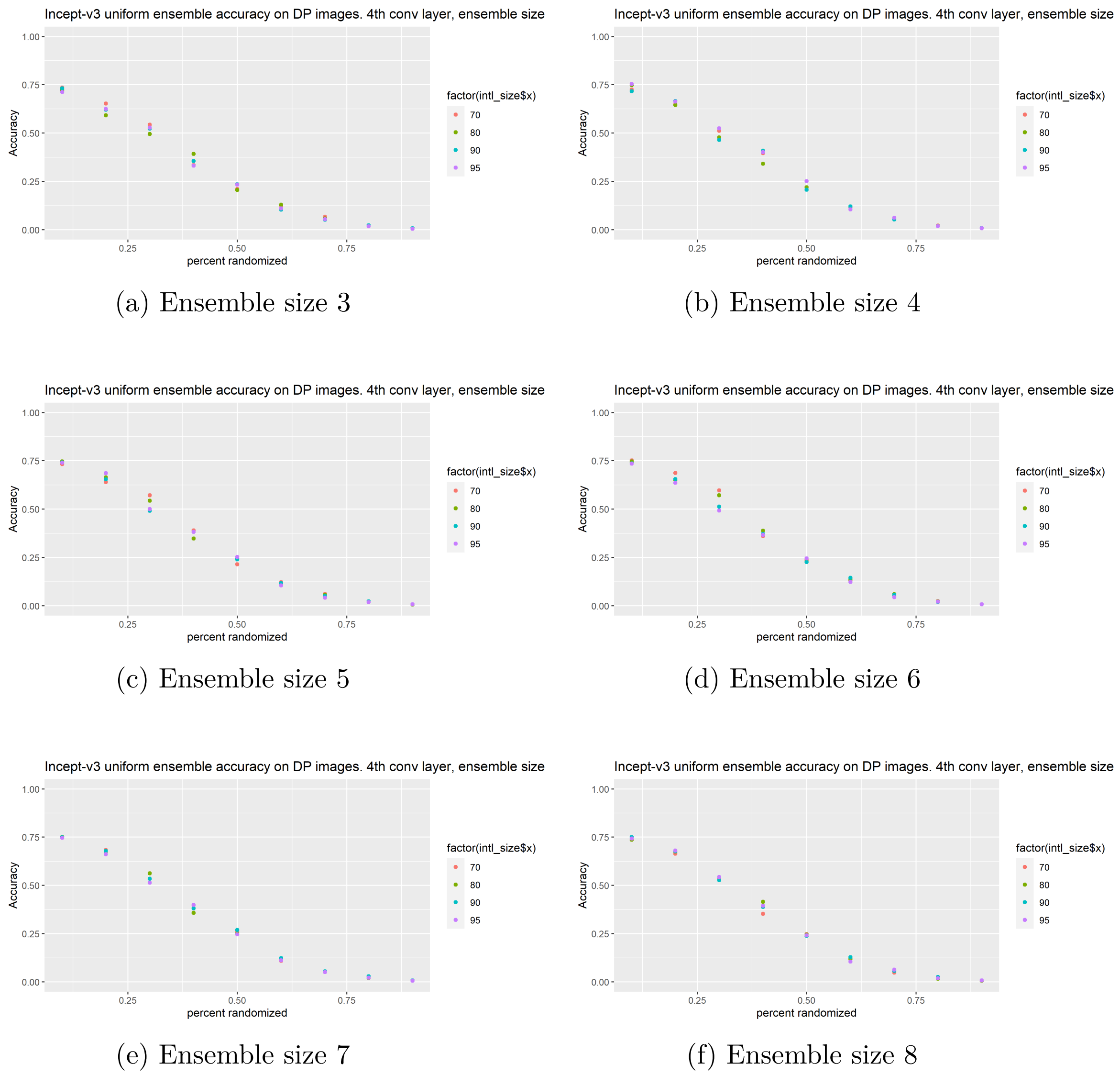
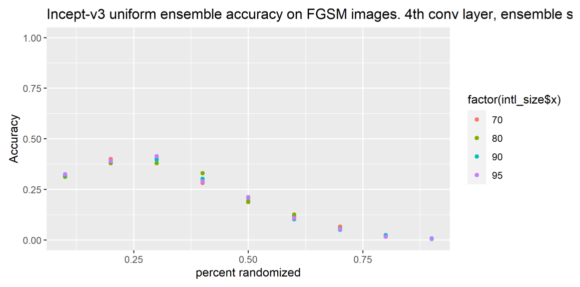
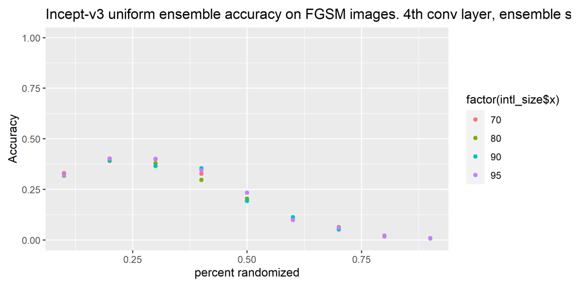


Figure 4.38. Ensemble results for Uniform randomization of Inception v3, fourth convolutional layer, on DeepFool attack images

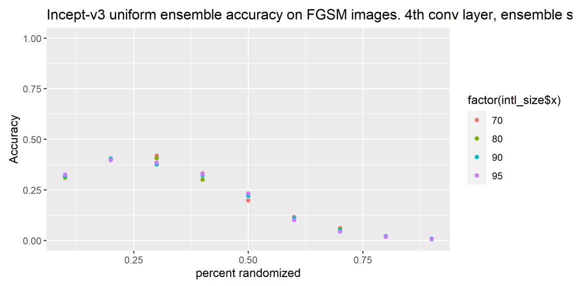
In Figure 4.38, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the fourth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the fourth convolutional layer. Ensemble size 3 has the highest accuracy 73.4%, with 70% interval. Ensemble size 4 has the highest accuracy 75.5%, with 95% interval. Ensemble size 5 has the highest accuracy 74.6%, with 80% interval. Ensemble size 6 has the highest accuracy 75.2%, with 70% interval. Ensemble size 7 has the highest accuracy 75.2%, with 80% interval. Ensemble size 8 has the highest accuracy 75.1%, with 90% interval.



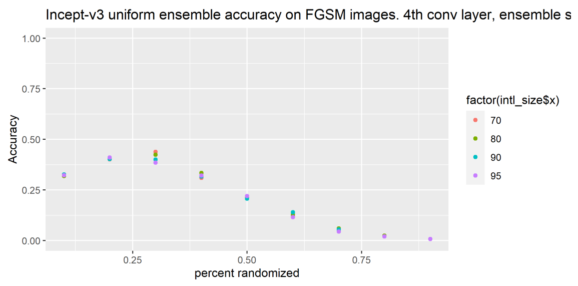
(a) Ensemble size 3



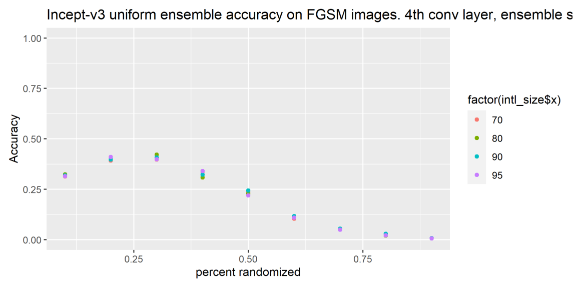
(b) Ensemble size 4



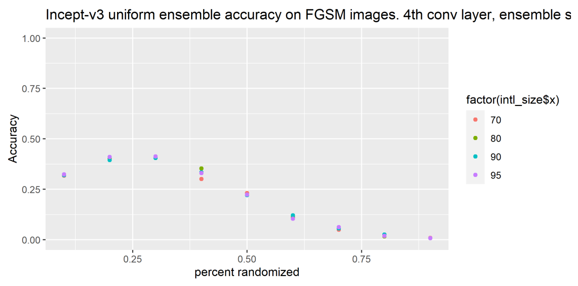
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

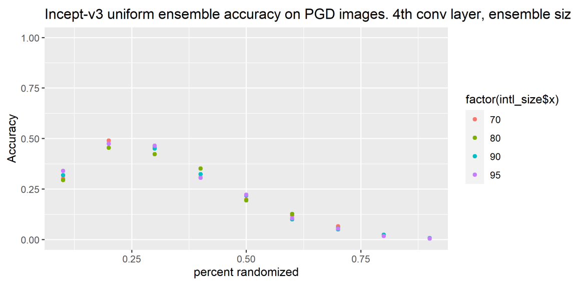


(f) Ensemble size 8

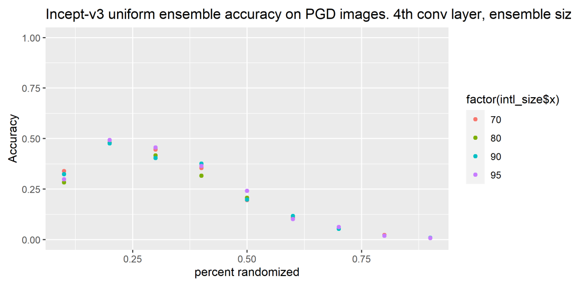
Figure 4.39. Ensemble results for Uniform randomization of Inception v3, fourth convolutional layer, on FGSM attack images

In Figure 4.39, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the fourth convolutional layer. For all ensemble sizes with the exception of ensemble size 4, the highest accuracy is achieved when randomizing 30% of the weights in the fourth convolutional layer. Ensemble size 3 has the highest accuracy 41.5%, with 95% interval. Ensemble size 4 has the highest accuracy 40.3%, with randomization of 20% of weights in fourth convolutional layer and 95% interval. Ensemble size 5 has the highest accuracy 41.9%, with 70% interval. Ensemble size 6 has the highest accuracy 43.9%, with 70% interval. Ensemble size 7 has the highest accuracy 42.2%, with 80% interval. Ensemble size 8 has the highest accuracy 41.3%, with 95% interval.

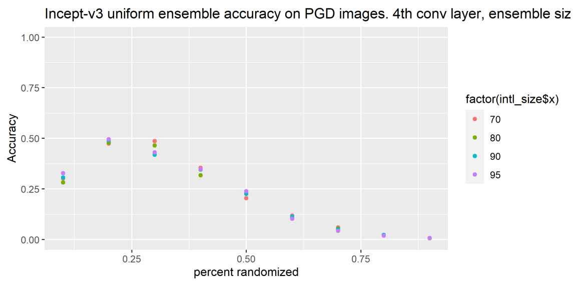
In Figure 4.40, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the fourth convolutional layer. For all ensemble sizes with the exception of ensemble size 6, the highest accuracy is achieved when randomizing 20% of the weights in the fourth convolutional layer. Ensemble size 3 has the highest accuracy 49.0%, with 70% interval. Ensemble size 4 has the highest accuracy 49.3%, with 95% interval. Ensemble size 5 has the highest accuracy 49.5%, with 95% interval. Ensemble size 6 has the highest accuracy 50.4%, with randomization of 30% of weights in fourth convolutional layer and 70% interval. Ensemble size 7 has the highest accuracy 50.2%, with 95% interval. Ensemble size 8 has the highest accuracy 50.9%, with 95% interval.



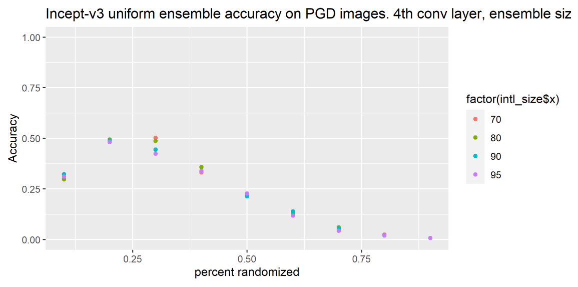
(a) Ensemble size 3



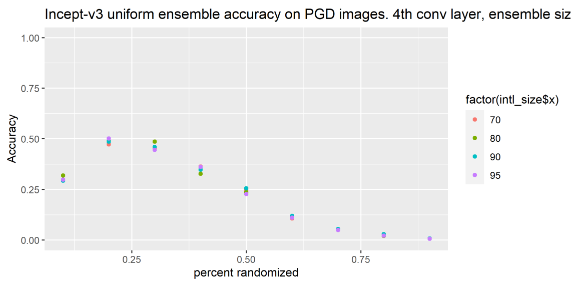
(b) Ensemble size 4



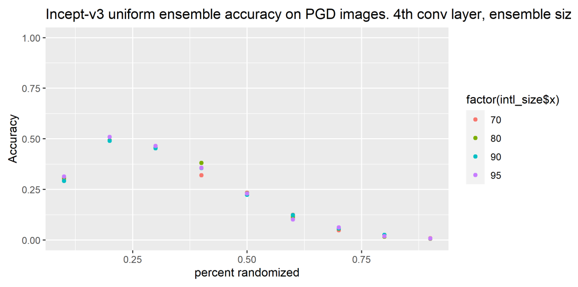
(c) Ensemble size 5



(d) Ensemble size 6

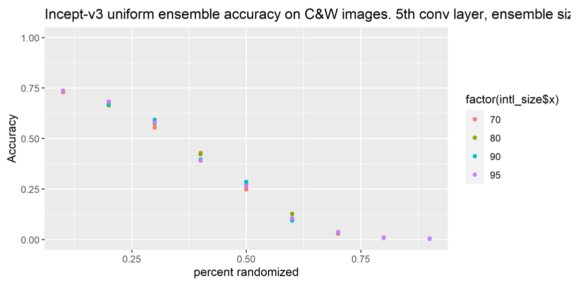


(e) Ensemble size 7

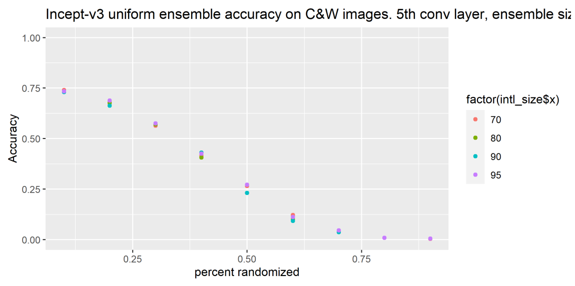


(f) Ensemble size 8

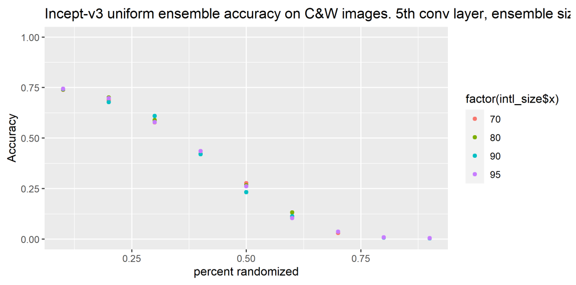
Figure 4.40. Ensemble results for Uniform randomization of Inception v3, fourth convolutional layer, on PGD attack images



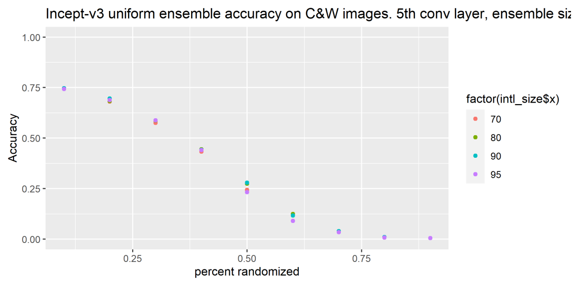
(a) Ensemble size 3



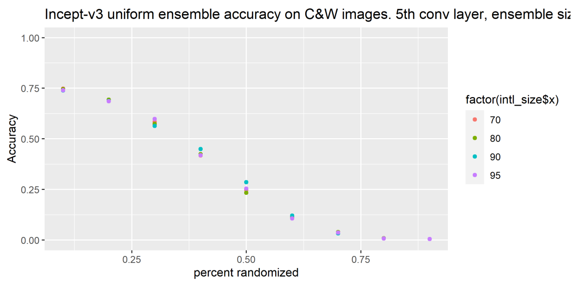
(b) Ensemble size 4



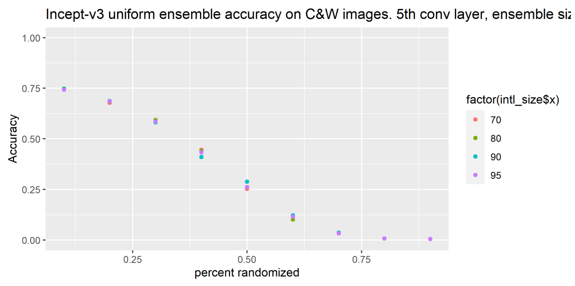
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

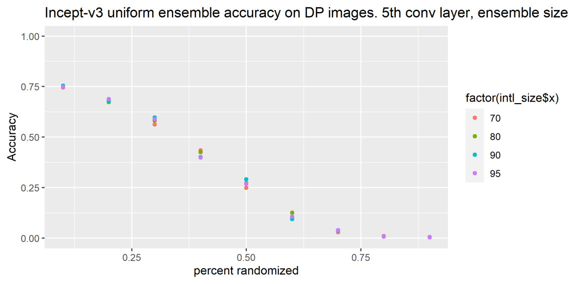


(f) Ensemble size 8

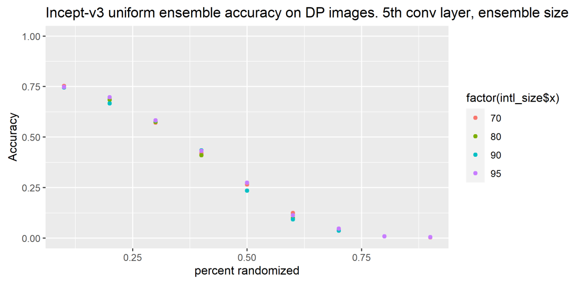
Figure 4.41. Ensemble results for Uniform randomization of Inception v3, fifth convolutional layer, on C&W attack images

In Figure 4.41, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the fifth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the fifth convolutional layer. Ensemble size 3 has the highest accuracy 73.7%, with 90% interval. Ensemble size 4 has the highest accuracy 74.1%, with 70% interval. Ensemble size 5 has the highest accuracy 74.5%, with 95% interval. Ensemble size 6 has the highest accuracy 74.6%, with 80% interval.

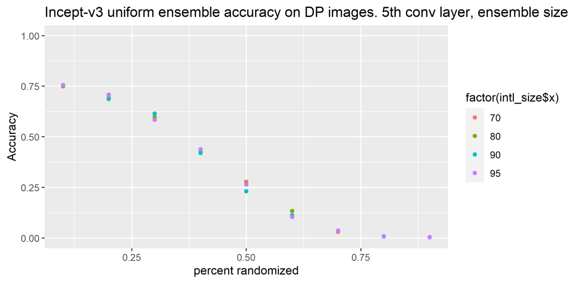
Ensemble size 7 has the highest accuracy 74.8%, with 70% interval. Ensemble size 8 has the highest accuracy 74.8%, with 70% interval.



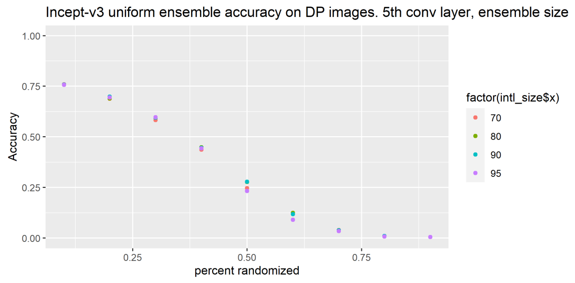
(a) Ensemble size 3



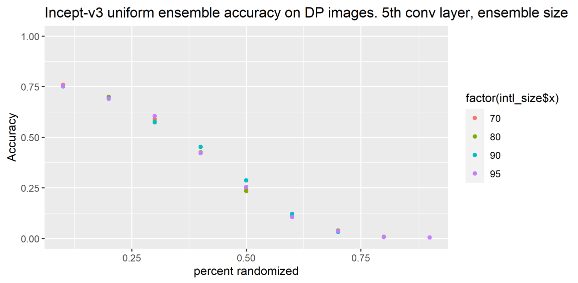
(b) Ensemble size 4



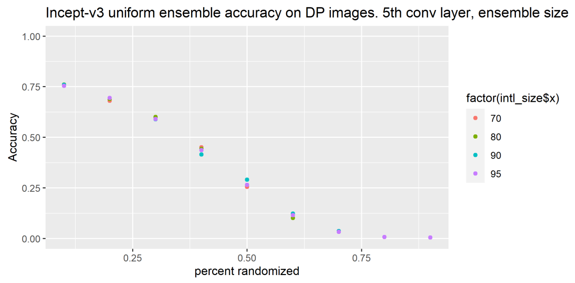
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

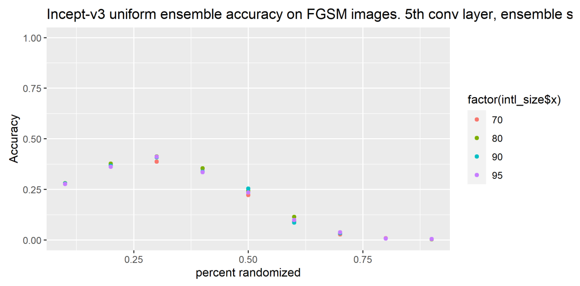


(f) Ensemble size 8

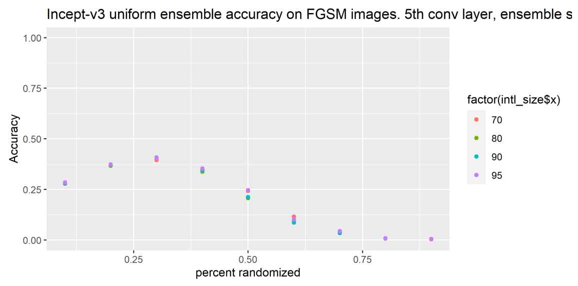
Figure 4.42. Ensemble results for Uniform randomization of Inception v3, fifth convolutional layer, on DeepFool attack images

In Figure 4.42, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the fifth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the fifth convolutional layer. Ensemble size 3 has the highest accuracy 75.4%, with 90% interval. Ensemble size 4 has the highest accuracy 75.3%, with 70% interval. Ensemble size 5 has the highest accuracy 75.6%,

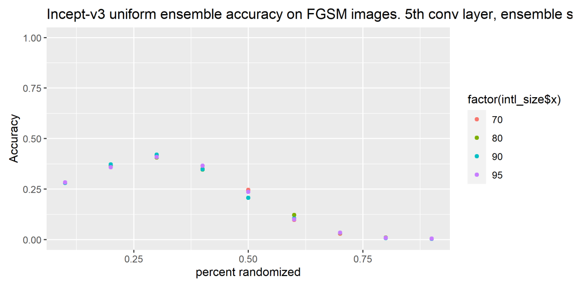
with 95% interval. Ensemble size 6 has the highest accuracy 76.0%, with 70% interval. Ensemble size 7 has the highest accuracy 75.9%, with 70% interval. Ensemble size 8 has the highest accuracy 76.1%, with 70% interval.



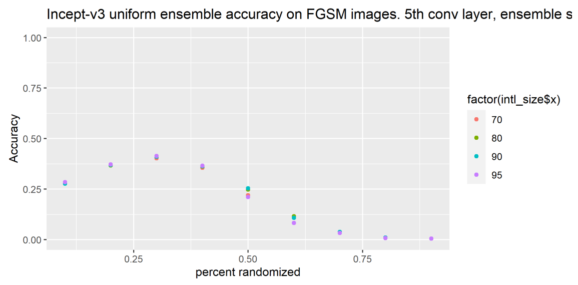
(a) Ensemble size 3



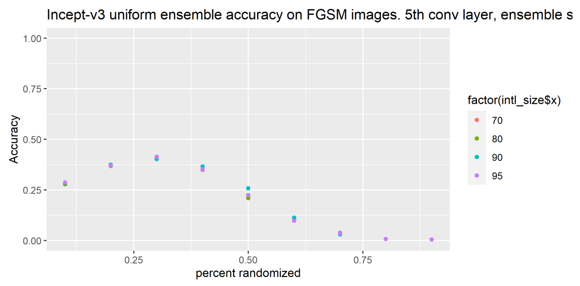
(b) Ensemble size 4



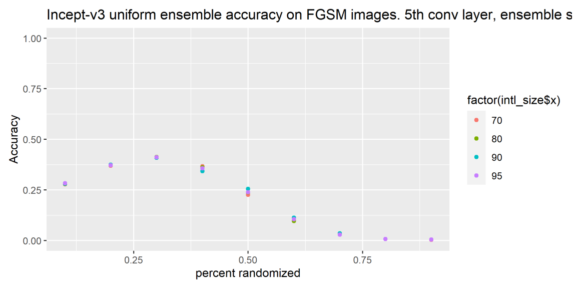
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

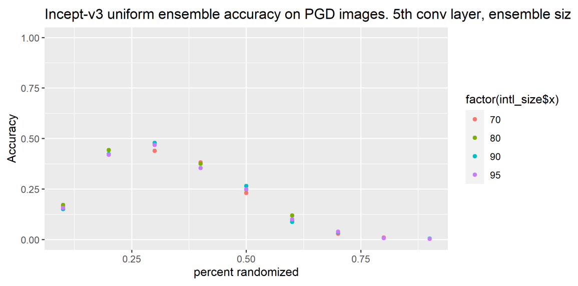


(f) Ensemble size 8

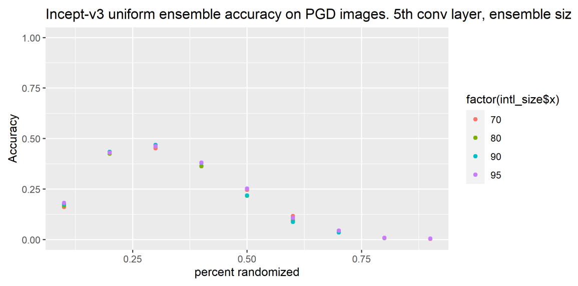
Figure 4.43. Ensemble results for Uniform randomization of Inception v3, fifth convolutional layer, on FGSM attack images

In Figure 4.43, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the fifth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 30% of the weights in the fifth convolutional layer. Ensemble size 3 has the highest accuracy 41.2%, with 80% interval. Ensemble size 4 has the

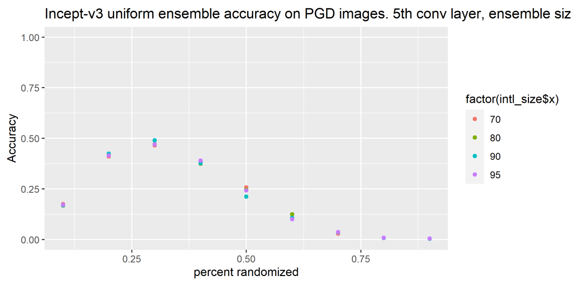
highest accuracy 40.8%, with 80% interval. Ensemble size 5 has the highest accuracy 42.1%, with 90% interval. Ensemble size 6 has the highest accuracy 41.4%, with 95% interval. Ensemble size 7 has the highest accuracy 41.5%, with 95% interval. Ensemble size 8 has the highest accuracy 41.3%, with 80% interval.



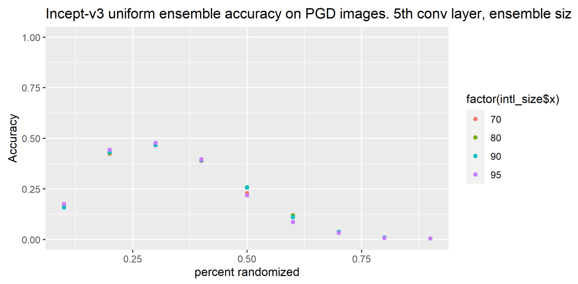
(a) Ensemble size 3



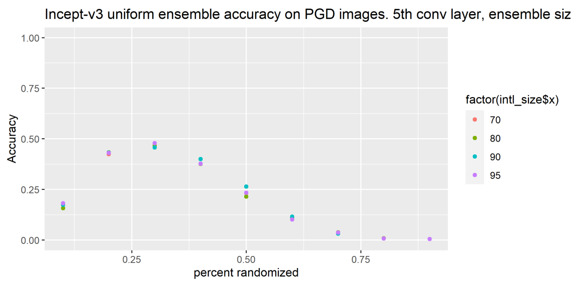
(b) Ensemble size 4



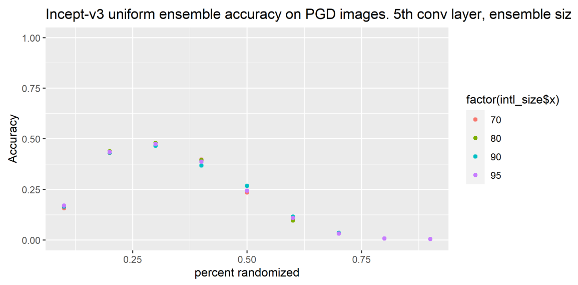
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7

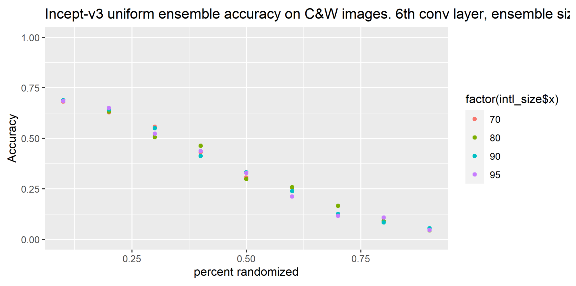


(f) Ensemble size 8

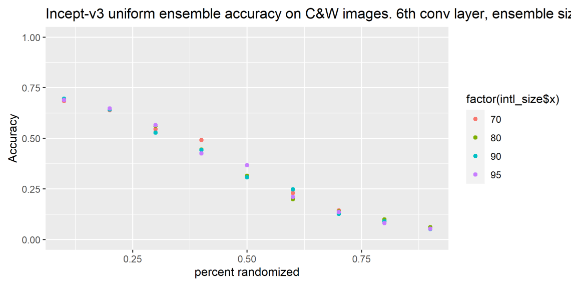
Figure 4.44. Ensemble results for Uniform randomization of Inception v3, fifth convolutional layer, on PGD attack images

In Figure 4.44, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the fifth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 30% of the weights in the fifth convolutional layer.

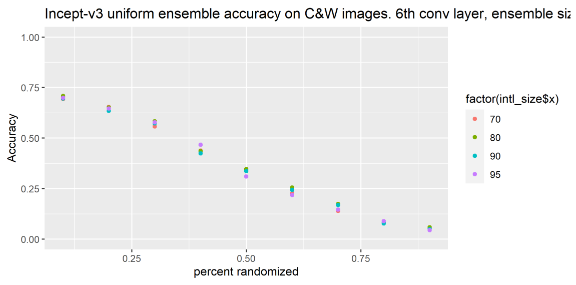
Ensemble size 3 has the highest accuracy 47.9%, with 90% interval. Ensemble size 4 has the highest accuracy 46.9%, with 90% interval. Ensemble size 5 has the highest accuracy 49.1%, with 90% interval. Ensemble size 6 has the highest accuracy 47.8%, with 95% interval. Ensemble size 7 has the highest accuracy 47.9%, with 95% interval. Ensemble size 8 has the highest accuracy 48.1%, with 80% interval.



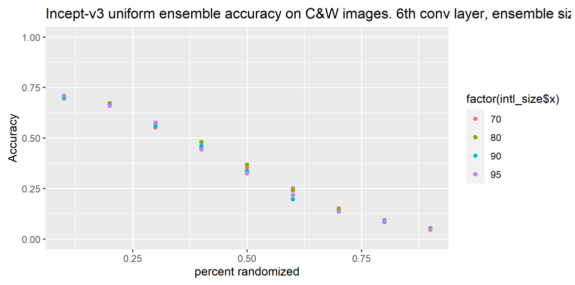
(a) Ensemble size 3



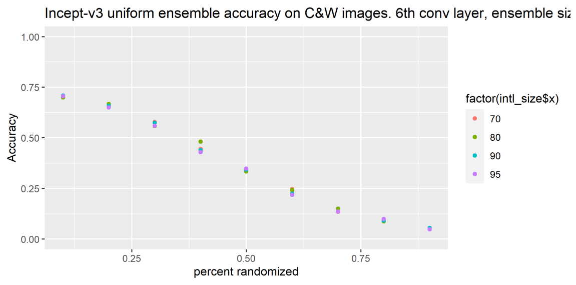
(b) Ensemble size 4



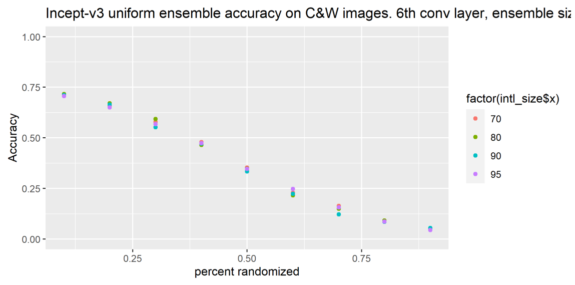
(c) Ensemble size 5



(d) Ensemble size 6



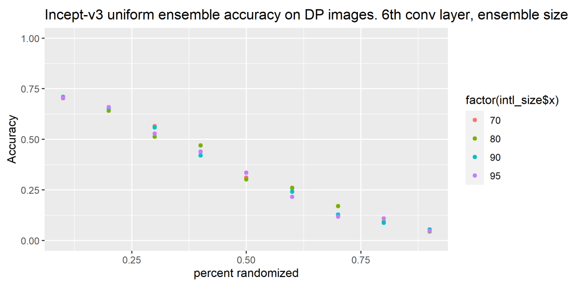
(e) Ensemble size 7



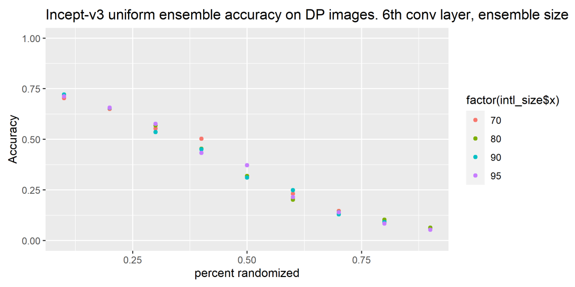
(f) Ensemble size 8

Figure 4.45. Ensemble results for Uniform randomization of Inception v3, sixth convolutional layer, on C&W attack images

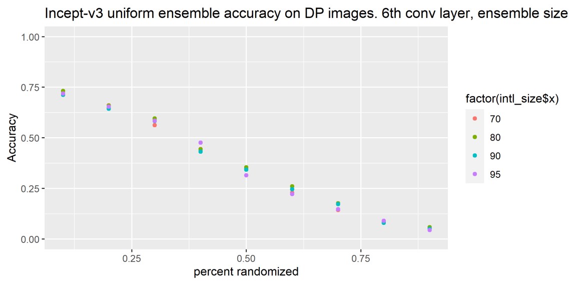
In Figure 4.45, we see accuracy on the C&W attack images with Inception v3 Uniform ensembles which randomize the sixth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the sixth convolutional layer. Ensemble size 3 has the highest accuracy 68.9%, with 90% interval. Ensemble size 4 has the highest accuracy 69.6%, with 90% interval. Ensemble size 5 has the highest accuracy 70.9%, with 80% interval. Ensemble size 6 has the highest accuracy 71.1%, with 95% interval. Ensemble size 7 has the highest accuracy 70.9%, with 70% interval. Ensemble size 8 has the highest accuracy 71.6%, with 80% interval.



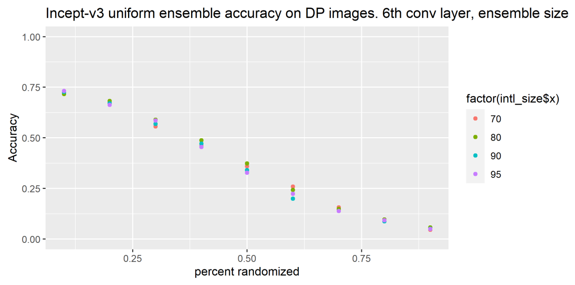
(a) Ensemble size 3



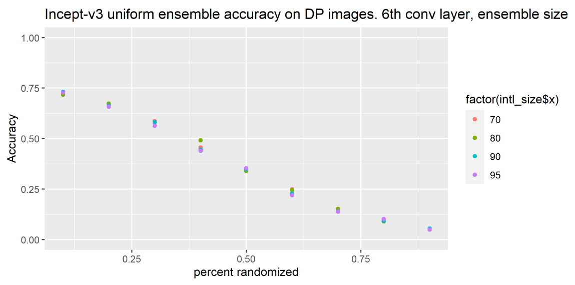
(b) Ensemble size 4



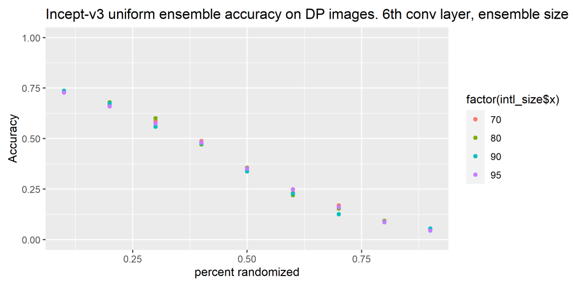
(c) Ensemble size 5



(d) Ensemble size 6



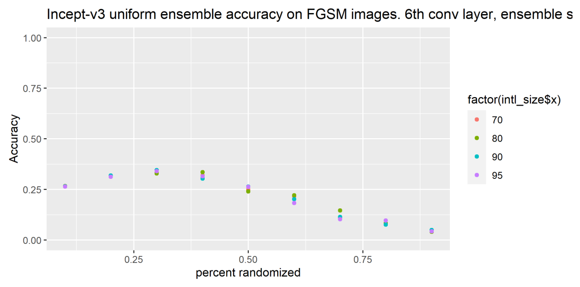
(e) Ensemble size 7



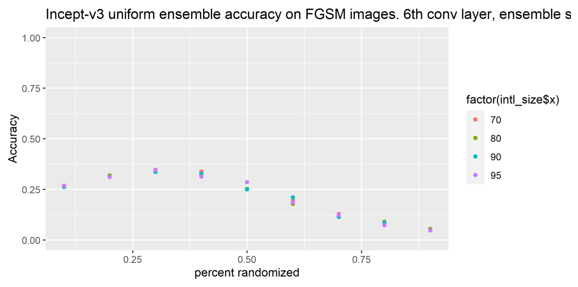
(f) Ensemble size 8

Figure 4.46. Ensemble results for Uniform randomization of Inception v3, sixth convolutional layer, on DeepFool attack images

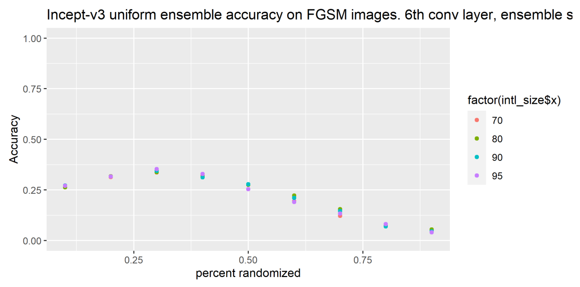
In Figure 4.46, we see accuracy on the DeepFool attack images with Inception v3 Uniform ensembles which randomize the sixth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 10% of the weights in the sixth convolutional layer. Ensemble size 3 has the highest accuracy 71.0%, with 90% interval. Ensemble size 4 has the highest accuracy 72.1%, with 90% interval. Ensemble size 5 has the highest accuracy 73.2%, with 80% interval. Ensemble size 6 has the highest accuracy 73.1%, with 95% interval. Ensemble size 7 has the highest accuracy 73.2%, with 70% interval. Ensemble size 8 has the highest accuracy 73.6%, with 90% interval.



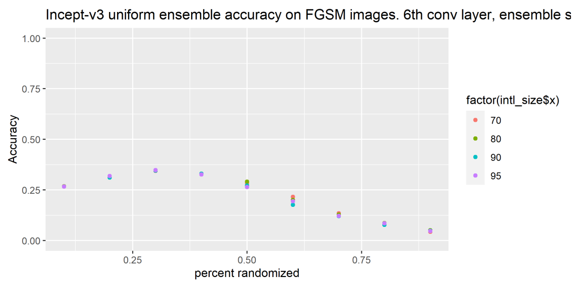
(a) Ensemble size 3



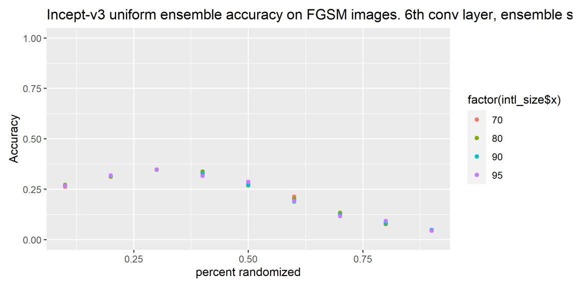
(b) Ensemble size 4



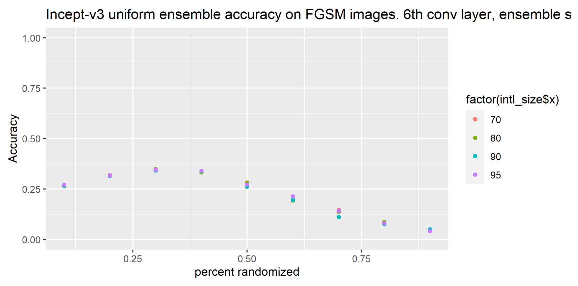
(c) Ensemble size 5



(d) Ensemble size 6



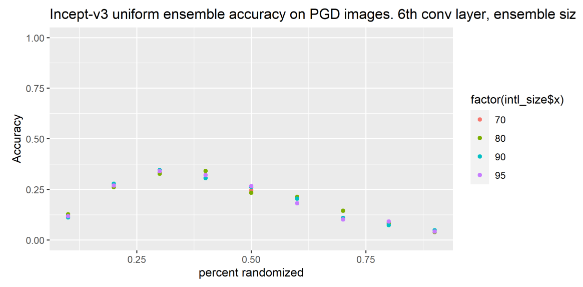
(e) Ensemble size 7



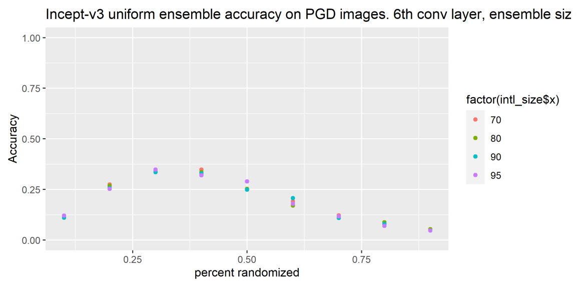
(f) Ensemble size 8

Figure 4.47. Ensemble results for Uniform randomization of Inception v3, sixth convolutional layer, on FGSM attack images

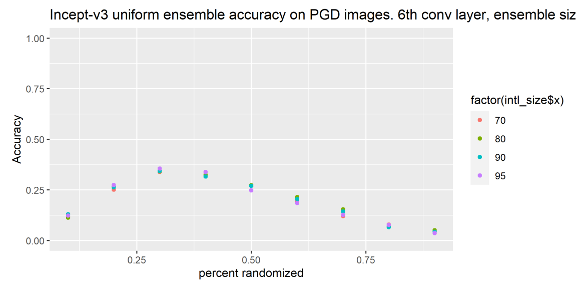
In Figure 4.47, we see accuracy on the FGSM attack images with Inception v3 Uniform ensembles which randomize the sixth convolutional layer. For all ensemble sizes, the highest accuracy is achieved when randomizing 30% of the weights in the sixth convolutional layer. Ensemble size 3 has the highest accuracy 34.5%, with 90% interval. Ensemble size 4 has the highest accuracy 34.6%, with 95% interval. Ensemble size 5 has the highest accuracy 35.4%, with 95% interval. Ensemble size 6 has the highest accuracy 34.9%, with 95% interval. Ensemble size 7 has the highest accuracy 34.9%, with 90% interval. Ensemble size 8 has the highest accuracy 34.8%, with 80% interval.



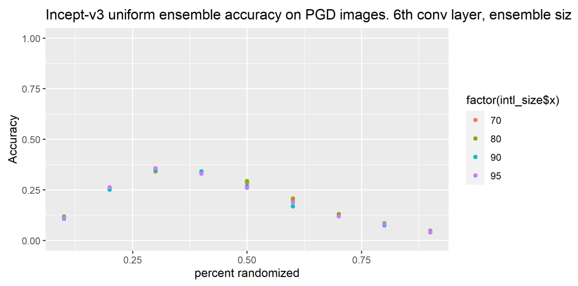
(a) Ensemble size 3



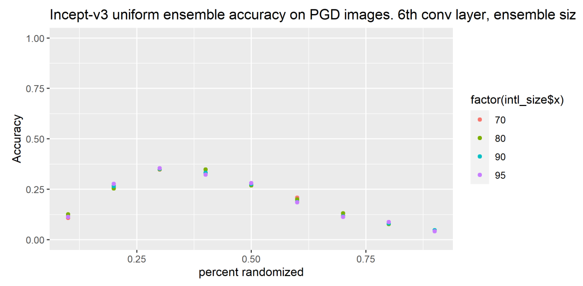
(b) Ensemble size 4



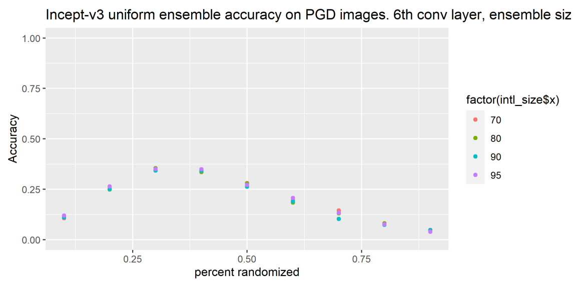
(c) Ensemble size 5



(d) Ensemble size 6



(e) Ensemble size 7



(f) Ensemble size 8

Figure 4.48. Ensemble results for Uniform randomization of Inception v3, sixth convolutional layer, on PGD attack images

In Figure 4.48, we see accuracy on the PGD attack images with Inception v3 Uniform ensembles which randomize the sixth convolutional layer. For all ensemble sizes with the exception of ensemble size 4, the highest accuracy is achieved when randomizing 30% of the weights in the sixth convolutional layer. Ensemble size 3 has the highest accuracy 34.6%, with 90% interval. Ensemble size 4 has the highest accuracy 34.9%, with randomization of 40% of weights in sixth convolutional layer and 70% interval. Ensemble size 5 has the highest accuracy 35.6%, with 95% interval. Ensemble size 6 has the highest accuracy 35.7%, with 95% interval. Ensemble size 7 has the highest accuracy 35.4%, with 95% interval. Ensemble size 8 has the highest accuracy 35.5%, with 80% interval.

4.7 Comparisons

We compare the performance of the random ensembles with two defense methods: adversarial training and Barrage of Random Transforms (BaRT) [72]. Adversarial training, the most popular defense strategy, is introduced in Chapter 2. BaRT is a defense strategy based on image pre-processing. Chapter 2 has a general discussion regarding using image pre-processing as a defense. BaRT includes ten image transformations: denoising, JPEG compression, image swirling, Fourier transform perturbations, noise injection, color space changes, zooming, histogram equalization, and grayscale transformations. For every batch of images, BaRT randomly selects five transformations and apply them sequentially to the images. Then the transformed images are handled by a DNN model. BaRT is a more sophisticated preprocessing defense.

Compared with BaRT and adversarial training, the best random ensembles have higher accuracy for CW2 attack and DeepFool attack. This happens to the different DNN models and image data-sets. The random ensembles have worse performance against PGD attack and FGSM, as in Tables 4.3 and 4.4. We compute the average amount of perturbations introduced by different attacks, in Table 4.2. DeepFool and CW2 attacks introduced the smallest amount of adversarial perturbations. PGD attack generate much larger adversarial perturbations. FGSM attack, one of the earliest, generate the largest amount of adversarial perturbation. One measure to evaluate the effectiveness of an attack is to measure the amount of adversarial perturbations. The attack that requires the smallest adversarial perturbation is considered a strong attack. In this sense, CW2 is a strong benchmark attack, where the random ensembles out-perform. The random ensembles become less accurate when very large adversarial perturbations are added to the samples.

The random ensembles of Inception V3 have very strong performance on Tiny ImageNet against all four attacks. Using random weights at the front of a DNN model has the best result for DNN with big and complex model structure.

Table 4.2. Average L2 distance between clean and adversarial examples.

Model	**PGD**	**FGSM**	**DeepFool**	**C&W2**
CNN3	9.999	13.088	5.379	4.411
VGG16	2.999	7.955	1.559	0.825
ResNet18	2.999	7.970	1.634	0.913
Incept v3	2.999	7.999	0.715	1.106

Table 4.3. Accuracy of BaRT defense on clean and adversarial images

Model	**Clean images**	**PGD**	**FGSM**	**DeepFool**	**C&W2**
CNN3	0.988	0.000	0.049	0.444	0.293
VGG16	0.865	0.592	0.439	0.820	0.834
ResNet18	0.806	0.545	0.373	0.750	0.764
Incept v3	0.721	0.105	0.166	0.316	0.188

Table 4.4. Accuracy of adversarial training defense on clean and adversarial images

Model	**Clean images**	**PGD**	**FGSM**	**DeepFool**	**C&W2**
CNN3	0.981	0.338	0.861	0.713	0.530
VGG16	0.880	0.631	0.417	0.800	0.836
ResNet18	0.806	0.584	0.398	0.731	0.750
Incept v3	0.748	0.589	0.498	0.635	0.713

5. CANCER DATA PROJECT

Ovarian cancer (OC) is the second most common form of gynecologic cancer and is the most fatal among all forms of gynecologic malignancies [73]. Despite the important role of metabolic processes in the molecular pathogenesis of OC, robust metabolic markers to enable effective screening, rapid diagnosis, accurate surveillance, and therapeutic monitoring of OC are still lacking. In this study, we present a targeted liquid chromatography-tandem mass spectrometry (LC-MS/MS)-based metabolic profiling approach for the identification of both lipid and metabolite biomarker candidates that could enable expedited, highly sensitive and specific OC detection. Using this targeted approach, 90 plasma metabolites from many metabolic pathways of potential biological significance were reliably detected and monitored in 218 plasma samples taken from three groups of subjects (78 OC patients, 50 benign samples, and 90 healthy controls). Univariate significance testing and receiver operating characteristic (ROC) analysis revealed 7 metabolites with high predictive accuracy [area under curve (AUC) > 0.90] for distinguishing healthy controls from OC patients. The results of our multivariate model development informed the construction of a 5-metabolite panel of potential plasma biomarkers for enhanced discrimination of OC samples from benign specimens, exhibiting roughly 75% predictive accuracy using a 50% random-split training set. ROC analysis that was generated based on a logistic regression classifier showed enhanced classification performance relative to individual metabolites, with more than 75% accuracy using a testing data set for external validation. Pathway analysis revealed significant disturbances in glycine, serine, and threonine metabolism; glyoxylate and dioxylate metabolism; the pentose phosphate pathway; and histidine metabolism. The results expand basic knowledge of the metabolome related to OC pathogenesis relative to healthy controls and benign samples, revealing potential pathways or markers that can be targeted therapeutically. This study also provides a promising basis for the development of larger multi-site projects to validate our findings across population groups and further advance the development of improved clinical care for OC patients.

It is reported that OC has an annual incidence rate of 239,000 and accounts for a total of 152,000 deaths each year, ranking it the seventh most common form of cancer among women worldwide.[74] Unlike other forms of cancers where there are recommended screening techniques available with high sensitivity to detect the early development of tumors, there are currently no effective routine screening tests recommended for OC.[75] Thus, due to either similarities in symptoms between OC and other diseases or lack of symptoms altogether, most cases are diagnosed in late stages of disease progression when 5-year survival rates are only 29%, whereas only a mere 15% of OC cases are diagnosed in stage 1 when 5-year survival rates are reported as high as 92%.[74] Current screening techniques used regularly in the clinical diagnosis of symptomatic OC, such as pelvic examination and transvaginal ultrasound, are invasive and have been found vastly inaccurate.[76], [77] To date, the cancer antigen 125 (CA-125) blood biomarker remains one of the most commonly used tests for symptomatic OC diagnosis owing to its nonintrusive sampling, but is reported to produce high rates of false-positive diagnoses and is nonspecific to OC, as it can be reflective of various other malignant and nonmalignant conditions ranging from pancreatic cancer to liver cirrhosis.[78], [79] Therefore, a critical demand exists for a noninvasive diagnostic method high in both sensitivity and specificity, which would allow for the prompt diagnosis and effective treatment of OC.

Recent efforts focused on advanced methods for OC detection have borne promising results. The human epididymis protein 4 (HE4), a glycoprotein overly expressed in forms of female reproductive cancers, and mesothelin (MES), an antigen on the mesothelium that is shown to be elevated in OC, have been found to increase the accuracy of the CA-125 blood biomarker test when used in conjunction with the Risk of Ovarian Malignancy Algorithm (ROMA).[80], [81] Additionally, analyses of genetic alterations, such as using comparison genome hybridization (CGH) with cDNA and mRNA, have found interesting amplifications and deletions in certain genes that suggest mechanisms of early stages of OC tumor development and potential targets for pharmacological interventions.[82], [83] Specifically, the BRCA1/2 gene has become an increasingly popular target of genetic testing, as it has been linked to both OC and breast cancer, although only 22% of OC cases have been characterized as hereditary.[C] Hence, these methods, among many others, have failed to assert narrow diagnostic accuracies to encompass the complexities of OC.

Metabolic disturbances, downstream of altered genetic and proteomic factors, ultimately contribute to tumor malignancy and are a signature hallmark of cancer,[84] providing rationale for discovery and validation of tumorigenic biomarkers and possible therapeutic targets.[85] Indeed, increased metabolic activity, specifically glucose uptake via the Warburg Effect, is a signature mark of cancer. Consequently, metabolomics, the study of comprehensive arrays of small molecule metabolites and their interactions in biological systems, holds great potential in cancer detection given its high sensitivity and specificity in detecting various aspects of cellular metabolism.[84], [86] Of specific note is the ability to investigate lipid metabolites, which have proven auspicious in studies investigating cancer metabolism due to their functions in cellular membrane maintenance, energy storage, and molecular signaling pathways. Furthermore, the aptitudes of metabolomics and lipidomics to test patients via noninvasive methods, such as urine or blood sampling, make these methods highly desirable tools for biomarker discovery and early diagnoses. Several studies have employed various mass-spectrometry based metabolomics approaches to analyze altered metabolic phenotypes and biological pathways associated with cancer, leading to significant progress in understanding cancer pathogenesis, developing advanced diagnostic methods, and identifying novel drug targets for clinical practice.[85], [87], [88] Previous studies investigating metabolic phenotypes suggestive of OC malignancies compared to healthy controls and benign tumors have employed an assortment of approaches, including high-performance liquid chromatography mass spectrometry (HPLC-MSMS), ultraperformance liquid chromatography mass spectrometry (UPLC-MS), and rapid resolution liquid chromatography mass spectrometry (RRLC-MS), from which significant alterations in metabolic profiles specific to OC have been reported.[76],[89]–[91] A few notable patterns have emerged in the phenotypic makeup of metabolites associated with OC, namely, alterations in metabolites related to amino acid metabolism, where elevated concentrations of known energy substrates such as glutamine and glutamate have been reported,[92]–[94] and in carbohydrate metabolism, where byproducts of anaerobic respiration such as lactate have been found to be decreased in aqueous samples[95] and elevated in tissue samples.[96] Thus, the production of novel biomarkers of disease from metabolomics-based research could be used as a diagnostic tool in the early detection of OC and differentiation of OC from benign specimens.

While results of metabolomics studies for improved OC diagnosis are encouraging,[86], [89]–[92], [95], [97] most of the previous studies investigating OC biomarkers for improved diagnosis have, to date, have focused on discernment of cancer patients from healthy controls, limiting the clinical applicability of such tests in distinguishing benign tumors from cancerous malignancies. As such, there remains a critical need to examine a broad array of metabolites for accurate identification of OC from both benign and control samples. In this study, a large-scale, targeted metabolomics approach in addition to advanced multivariate statistics was applied to the analysis of OC, benign, and control samples for the purposes of biomarker discovery.

5.1 Experiment

5.1.1 Reagents

Acetonitrile (ACN), methanol (MeOH), ammonium acetate (NH4OAc), and acetic acid (AcOH), all LC-MS grade, were purchased from Fisher Scientific (Pittsburgh, PA). Ammonium hydroxide (NH4OH) was bought from Sigma-Aldrich (Saint Louis, MO). DI water was provided in-house by a Water Purification System from EMD Millipore (Billerica, MA). Phosphate buffered saline (PBS) was bought from GE Healthcare Life Sciences (Logan, UT). Standard compounds corresponding to the measured metabolites were purchased from Sigma-Aldrich (Saint Louis, MO) and Fisher Scientific (Pittsburgh, PA).

5.1.2 Clinical samples

The samples were collected under a previously approved IRB protocol with waived consent. Clinical samples were purchased from the Fred Hutchinson Cancer Research Center (FHCRC; Seattle, WA) and Bloodworks Northwest (Seattle, WA). Informed consent was obtained from all participants (OC patients, patients with benign samples, and healthy controls) before sample collection at the aforementioned research institutes. All participants were evaluated, and blood samples were obtained following an overnight fast. De-identified aliquots were provided to the Arizona Metabolomics Laboratory (College of Health Solutions, Arizona State University) for processing.

5.1.3 Sample preparation

Samples had been frozen under –80°C prior to analysis. Frozen plasma samples were first thawed overnight under 4°C. Afterward, 50 μL of each plasma sample was placed in a 2 mL Eppendorf vial. The initial step for protein precipitation and metabolite extraction was performed by adding 500 μL MeOH and 50 μL internal standard solution (containing 1,810.5 μM $^{13}C_3$-lactate and 142 μM $^{13}C_5$-glutamic acid). The mixture was then vortexed for 10 s and stored at –20°C for 30 min, followed by centrifugation at 14,000 RPM for 10 min at 4°C. The supernatants (450 μL) were collected into new Eppendorf vials and dried using a CentriVap Concentrator (Labconco, Fort Scott, KS). The dried samples were reconstituted in 150 μL of 40% PBS/60% ACN and centrifuged again at 14,000 RPM at 4°C for 10 min. After that, 100 μL of supernatant was collected from each sample into an LC autosampler vial for subsequent analysis. A pooled sample, which was a mixture of all plasma samples, were used as the internal quality-control (QC) sample and injected once every 10 experimental samples.

The targeted LC-MS/MS method used here was modeled after that developed and used in a growing number of studies. [98]–[103] Briefly, all LC-MS/MS experiments were performed on an Agilent 1290 UPLC-6490 QQQ-MS system (Santa Clara, CA). Each sample was injected twice, 10 μL for analysis using negative ionization mode and 4 μL for analysis using positive ionization mode. Both chromatographic separations were performed in hydrophilic interaction chromatography (HILIC) mode on a Waters XBridge BEH Amide column (150 x 2.1 mm, 2.5 μm particle size, Waters Corporation, Milford, MA). The flow rate was 0.3 mL/min, auto-sampler temperature was kept at 4°C, and the column compartment was set to 40°C. The mobile phase was composed of Solvents A (10 mM ammonium acetate, 10 mM ammonium hydroxide in 95% H_2O/5% ACN) and B (10 mM ammonium acetate, 10 mM ammonium hydroxide in 95% ACN/5% H_2O). After an initial 1 min isocratic elution of 90% B, the percentage of Solvent B decreased to 40% at t = 11 min. The composition of Solvent B was maintained at 40% for 4 min (t = 15 min), after which the percentage of B gradually returned to 90%, in preparation for the next injection.

The mass spectrometer was equipped with an electrospray ionization (ESI) source. Targeted data acquisition was performed in multiple-reaction-monitoring (MRM) mode. We monitored 118 and 160 MRM transitions in negative and positive mode, respectively (278 transitions in total). The whole LC-MS system was controlled by Agilent MassHunter Workstation software (Santa Clara, CA). The extracted MRM peaks were integrated using Agilent MassHunter Quantitative Data Analysis software (Santa Clara, CA).

5.2 Preliminary analysis

Univariate testing was performed using SPSS 22.0 (SPSS Inc., Chicago, IL). Multivariate statistical analyses were performed using open-source R software and SIMCA-P (Umetrics, Umeå, Sweden). The data were $\log_{10}$-transformed prior to significance testing and model construction. Pathway analysis and integrating enrichment analysis were performed and visualized using the online MetaboAnalyst software.[104]

A total of 78 OC patients, 90 healthy controls, and 50 benign samples were included in the study. There was no statistically significant difference in age ($p > 0.05$) between OC, control, and benign groups as calculated by the Mann-Whitney U test. Table 5.1 shows the clinical characteristics of subjects included in the study.

Table 5.1. Clinical and characteristics of study subjects.

	Ovarian Cancer	**Benign**	**Healthy Control**
N	78	50	90
Age (mean±SD)	66.88±8.49	63.57±9.72	65.37±10.63
Stage I and II	56 (72%)		
Stage III and IV	22 (28%)		
Serous cystadenoma		14 (28%)	
Ovary		8 (16%)	
Leiomyomata		8 (16%)	
Mature cystic teratoma		5 (10%)	
Others		15 (30%)	

In the current study, we used a large-scale, targeted LC-MS/MS approach for reliable and comprehensive OC plasma metabolic profiling.[87] Using this metabolic profiling system, targeted analysis of 278 MRM transitions was achieved for metabolites spanning over 20 different chemical classes (such as amino acids, carboxylic acids, pyridines, etc.) from more than 35 metabolic pathways (e.g., TCA cycle, amino acid metabolism, glycolysis, purine and pyrimidine metabolism, urea cycle, etc.) in both positive and negative ionization modes. In total, we found that 90 plasma metabolites were reliably detected with relative abundances $>$ 1,000 in more than 80% of all samples. After normalization by averaged values from QC injection data, relative levels of the 90 plasma metabolites had a median coefficient of variation (CV) value of 11.53% (range: 3.37%-19.95%) with 75% of metabolites having CV $<$ 15%.

Of the 90 reliably detected plasma metabolites, 32 showed statistical significance between OC patients, benign samples, and healthy controls as determined by Kruskal-Wallis one-way ANOVA testing. Volcano plots of the tested plasma metabolites showing significance and fold change values for each post-hoc comparison is presented in Figure 5.1. Comparatively, 24 metabolites were found to be significant and have a high degree of fold-change (FC) between cancer/healthy samples, while 20 metabolites met these criteria when comparing benign/healthy samples. For comparison of cancerous and benign samples, 5 metabolites had $p<0.05$ and FC$>$2 or $<$0.50. The 5 significant metabolites found to have a high magnitude of fold change were 4-pyridoxic acid, azelaic acid, biotin, sorbitol, and glycine.

To further explore potential biomarkers for discrimination between OC patients, non-OC benign samples, and healthy controls, levels of the 90 reliably detected plasma metabolites were subjected to receiver operating characteristic (ROC) analysis and results summarized in Table 5.2. When comparing healthy and benign samples, 5 metabolites had area under curve (AUC) $>$ 0.90, whereas 6 metabolites had AUC $>$ 0.90 for the comparison of healthy controls and OC patients. However, for discernment of benign and OC samples, no metabolites satisfied this criterion, with the highest metabolites showing 70% accuracy. In an effort to increase the predictive accuracy and meet an important clinical need, logistic regression was applied using levels of the 5 significant metabolites found to have a high magnitude of fold change [4-pyridoxic acid, azelaic acid, biotin, sorbitol, glycine]. Benign and cancer samples were randomly assigned to either a training set (50%) or testing set (50%) for external cross

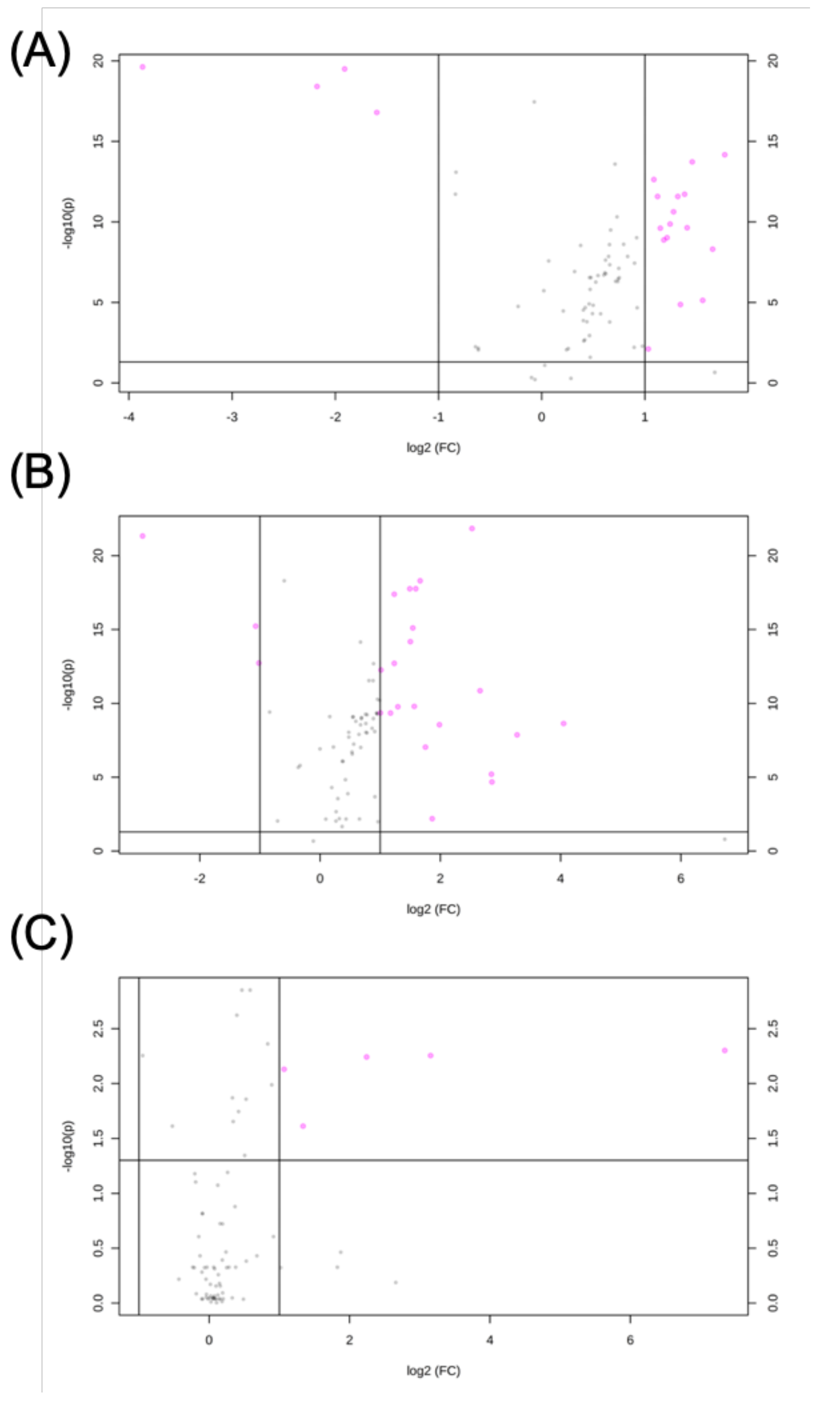


Figure 5.1. Volcano plot of (A) benign/healthy comparison, (B) cancer/healthy comparison, and (C) cancer/benign comparison. Fold change (FC) threshold: 2.0; FDR-adjusted p-value threshold: 0.05. Unequal group variance was assumed, non-parametric test was used.

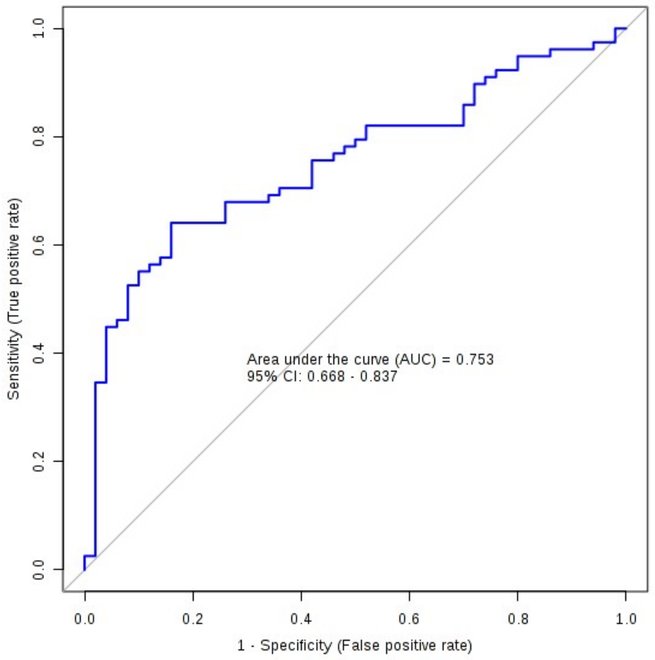


Figure 5.2. Evaluation of logistic regression model performance constructed using a 50% testing data set of benign and cancer samples [AUC = 0.753, sensitivity = 0.641 when specificity = 0.840].

validation. Construction of the logistic model using testing set data is given in Equation 5.1.

$$\begin{aligned}\text{Logit}(p) = -26.104 + 0.31 * \text{4-Pyridoxic acid} + 0.619 * \text{Azelaic acid} \\ + 0.194 * \text{Biotin}) + 0.26 * \text{Sorbitol} + 0.557 * \text{Glycine}\end{aligned} \tag{5.1}$$

Performance of the logistic training model was evaluated using ROC analysis. The model showed improved accuracy for discrimination of cancer from benign samples in comparison to the univariate performance of any individual metabolite [AUC = 0.791, sensitivity = 0.638 when specificity = 0.838]. The training set derived algorithm (Equation 5.1) was applied to testing set data in order to evaluate external validity and predictive performance. As depicted in Figure 5.2, the model proved to be robust when evaluating new samples, demonstrating predictive accuracy still higher than that of univariate classifiers [AUC = 0.753, sensitivity = 0.641 when specificity = 0.840].

Table 5.2. Univariate Metabolite Performance Based on Healthy vs Benign

Comparison	Metabolite	AUC
Healthy vs. Benign	N'N-Dicyclohexylurea	0.996
	Carnitine	0.991
	Creatine	0.952
	Acetylcholine	0.914
	Biotin	0.907
Healthy vs. Cancer	Biotin	0.957
	N'N-Dicyclohexylurea	0.949
	HIAA	0.914
	Kynurenine	0.906
	Acetylcholine	0.906
	9-Octadecynoic acid	0.901

Levels of the 5 significant metabolites with high fold factor change [4-pyridoxic acid, azelaic acid, biotin, sorbitol, glycine] to detect cancer from benign are shown as box plots in Figure 5.3. Unfortunately both the univariate selection and the selection based on fold change does not work sufficiently well for separating cancer from benign. Next we introduce a likelihood ratio biomarker selection method.

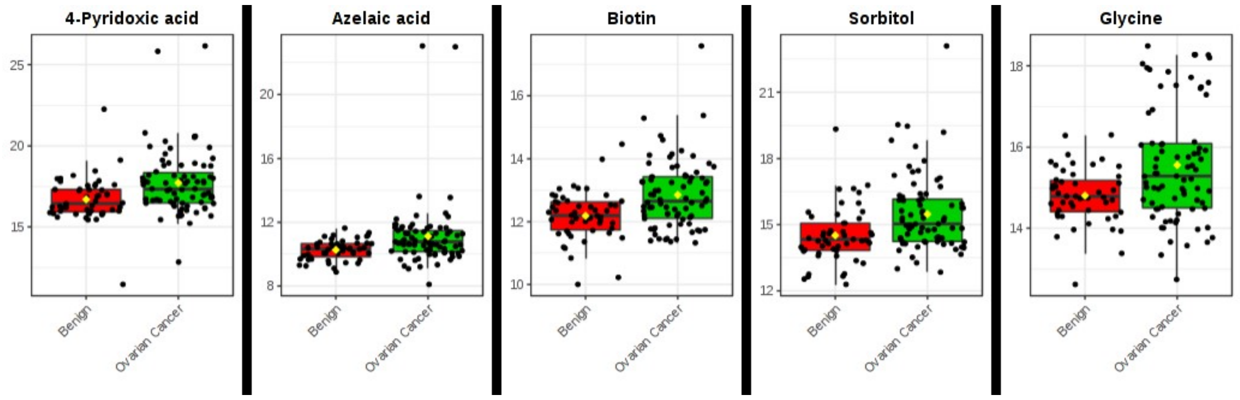


Figure 5.3. Box plots of candidate plasma markers (all $p < 0.05$ and FC > 2 or < 0.50) for detection of cancer from benign samples. Data were $\log_{10}$ normalized.

5.3 Metabolite selection using likelihood ratio approach

A likelihood ratio approach was implemented to discover additional metabolites of greatest statistical significance to be used in further analysis. This approach was implemented on pareto-scaled data from the UA 2 testing site. A test statistic based on the likelihood ratio was assigned to each metabolite, as the quotient of the likelihood of a bimodal distribution and the likelihood of a unimodal distribution. The test statistic, R_{j} , for the jth metabolite is the following.

$$R_{\mathrm{j}} = \frac{L[\,\hat{\mu_1}, \hat{\mu_2}, \hat{\sigma_1^2}, \hat{\sigma_2^2}\,|\,n_1, x^1_{1,\mathrm{j}}, ..., x_{n_1,\mathrm{j}}1, n_2, x^2_{1,\mathrm{j}}, ..., x^2_{n_2,\mathrm{j}}]}{L[\,\hat{\mu}, \hat{\sigma^2}\,|\,n, x^1_{1,\mathrm{j}}, ..., x^1_{n_1,\mathrm{j}}, x^2_{1,\mathrm{j}}, ..., x^2_{n_2,\mathrm{j}}]} \tag{5.2}$$

The plot of the test statistics is shown in Figure 5.4. Natural breaks in the plot can be seen at several points, including 8, 12, and 24 markers. The 12 metabolites with the highest likelihood ratio test statistic, in order, are PC (18:1/18:1), Oleic acid, PC 34:1, PC 34:2, PC 36:4, L-Alloisoleucine/Leucine/Norleucine, PC 34:0, Palmitic acid, PC 38:4, Acetylcarnitine, Betaine, and Isoleucine. Table 5.3 outlines all metabolites used in further analyses including Support Vector Machine (SVM) and Convolutional Neural Networks (CNNs).

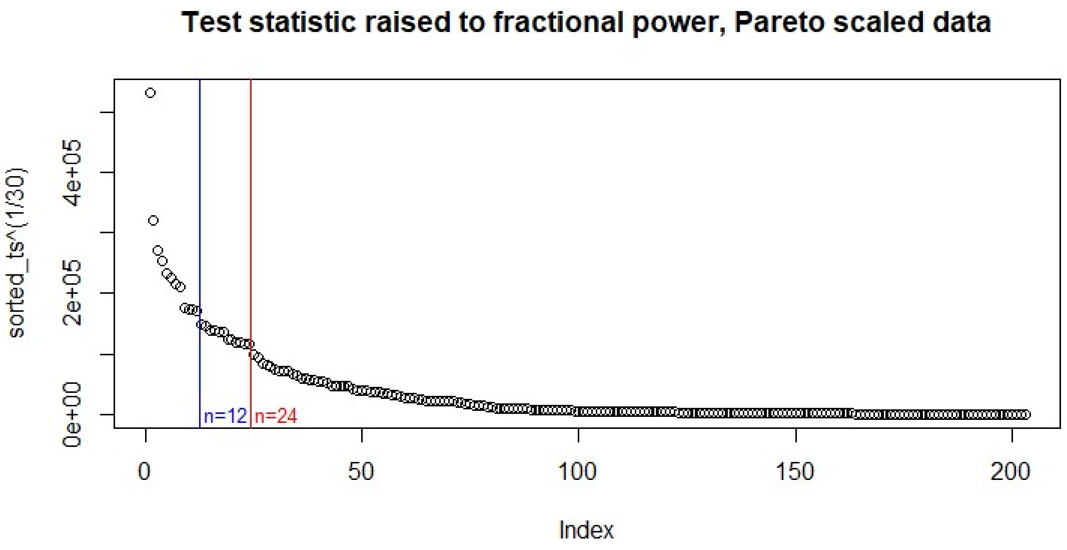


Figure 5.4. Selected biomarkers from both fold change and likelihood ratio approach.

Table 5.3. Selected Metabolites Based on Likelihood Ratio and Fold Change

Selection strategy	Biomarker	Biomarker type
Fold change	4-pyridoxic acid	Metabolite
	Azelaic acid	Metabolite
	Biotin	Metabolite
	Sorbitol	Metabolite
	Glycine	Metabolite
Likelihood ratio	PC (18:1/18:1)	Lipid
	Oleic acid	Lipid
	PC 34:1	Lipid
	PC 34:2	Lipid
	PC 36:4	Lipid
	L-Alloisoleucine/Leucine/Norleucine	Metabolite
	PC 34:0	Lipid
	Palmitic acid	Lipid
	PC 38:4	Lipid
	Acetylcarnitine	Metabolite
	Betaine	Metabolite
	Isoleucine	Metabolite

5.4 SVM accuracies and ROC

SVM was performed using R for Pareto scaled AML Ovarian Cancer data. The five markers identified by fold change analysis were used in all models. Additional markers were determined using the likelihood ratio approach on Pareto scaled data from testing site UA2. Up to 12 additional metabolites were considered when developing SVM models. A cut-off of 12 was selected based on a natural break visible in Figure 5.4. Different SVM models were run for several different subsets of the data. The data from UA 2 was considered to be the training dataset. Leave-one-out cross validation and radial kernel were used.

The highest level of validation accuracy in the training set is achieved when all 17 metabolites are in the model, with 77.2% accuracy. For all future analyses, the full 17 metabolites were considered in each model. Several group combinations of the dataset were considered, summarized in Table 5.4. The lowest SVM validation accuracies are achieved when comparing the cancerous and benign, regardless of cancer site. Accuracies above 90% were achieved when comparing Cancer and Healthy (regardless of cancer site), and both cancer sites combined compared to a combination of healthy and benign. ROC curves for each of the five two-way comparisons are shown in Figures 5.5 - 5.9.

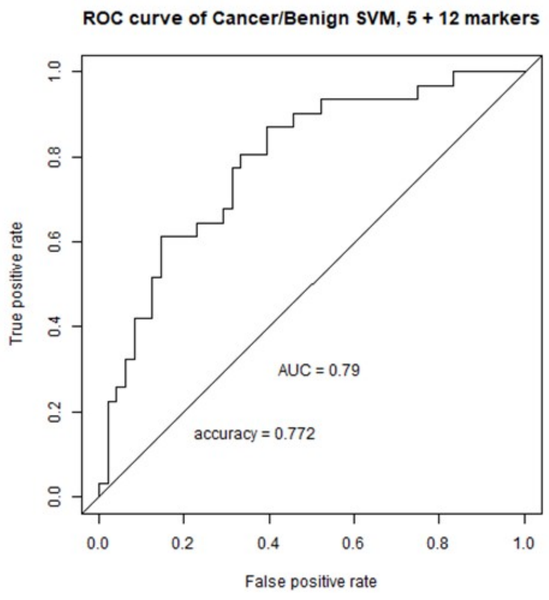


Figure 5.5. ROC curve of Cancer/Benign CNN, 17 markers.

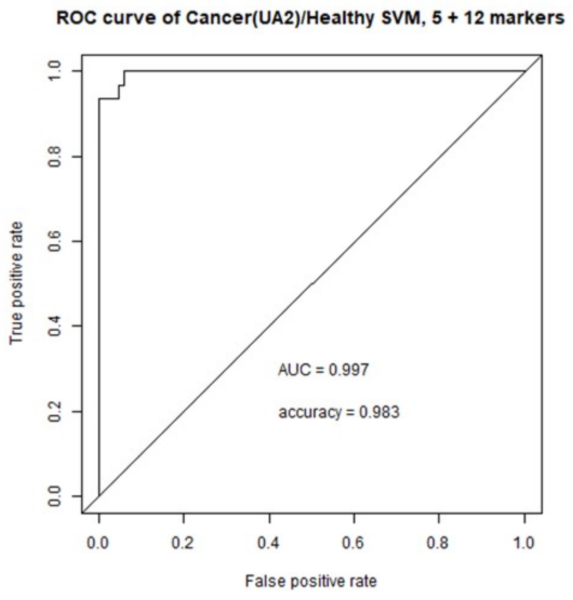


Figure 5.6. ROC curve of Cancer(UA2)/Healthy SVM, 17 markers.

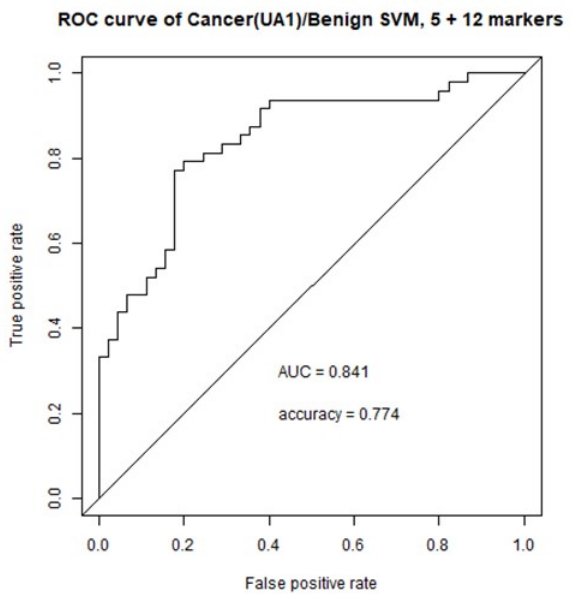


Figure 5.7. ROC curve of Cancer(UA1)/Benign SVM, 17 markers.

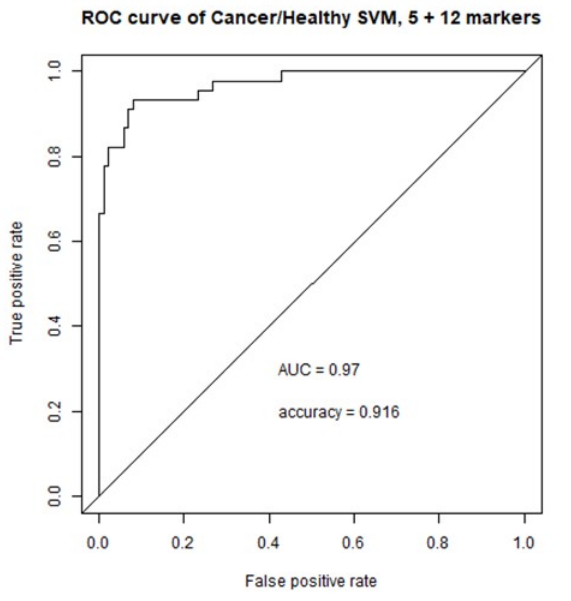


Figure 5.8. ROC curve of Cancer/Healthy SVM, 17 markers.

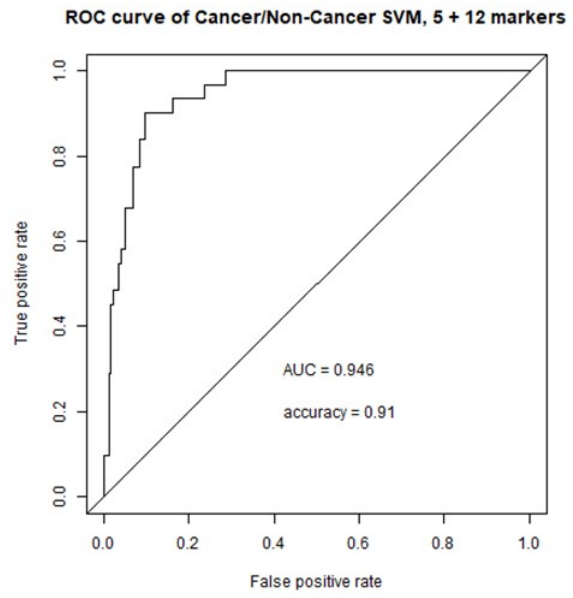


Figure 5.9. ROC curve of Cancer/Non-Cancer SVM, 17 markers.

Table 5.4. SVM validation accuracies. All accuracies are shown for models including 17 biomarkers.

Comparison Groups	**n**	**accuracy**
Cancer(UA2)/Benign	79	0.772
Cancer(UA2)/Healthy	117	0.983
Cancer(BioIVT)/Benign	93	0.774
Cancer(BioIVT)/Healthy	131	0.916
Cancer(UA2)/Benign/Healthy	165	0.8364
Cancer/Non-cancer	210	0.9095

5.5 CNN accuracies and ROC

Neural networks were used to classify the data. Models were created using 17 metabolites, including the 5 identified by fold change and the top 12 identified by the likelihood ratio approach. There were also networks trained using all 203 biomarkers. The following models were run in Matlab using trainNetwork. The accuracies listed in Table 5.5 are that of leave-one-out cross validation. A single layer convolutional neural network (CNN) was employed with an initial learn rate of 1e-5.

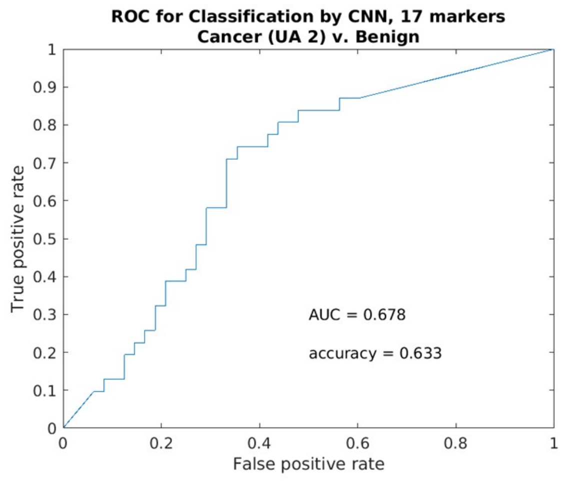


Figure 5.10. ROC curve of Cancer/Benign CNN, 17 markers.

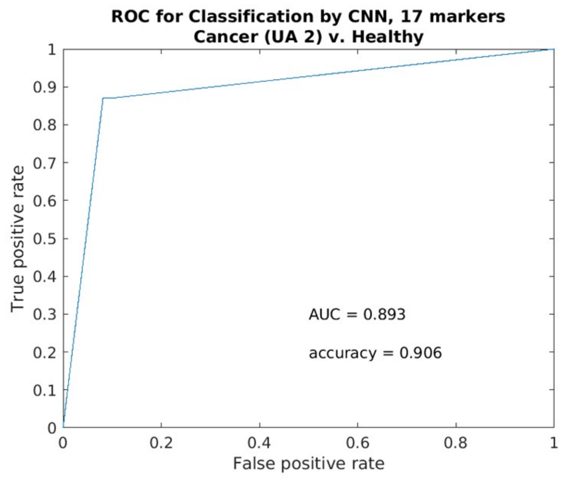


Figure 5.11. ROC curve of Cancer(UA2)/Healthy CNN, 17 markers.

Table 5.5. CNN leave-one-out cross validation accuracies.

Comparison Groups	**n**	**17 markers**	**203 markers**
Cancer(UA2)/Benign	79	0.6329	0.5443
Cancer(UA2)/Healthy	117	0.9060	0.5470
Cancer(BioIVT)/Benign	93	0.6452	0.4731
Cancer(BioIVT)/Healthy	131	0.5267	0.4962
Cancer(UA2)/Benign/Healthy	165	0.5818	0.3394
Cancer/Non-cancer	210	0.5476	0.5095

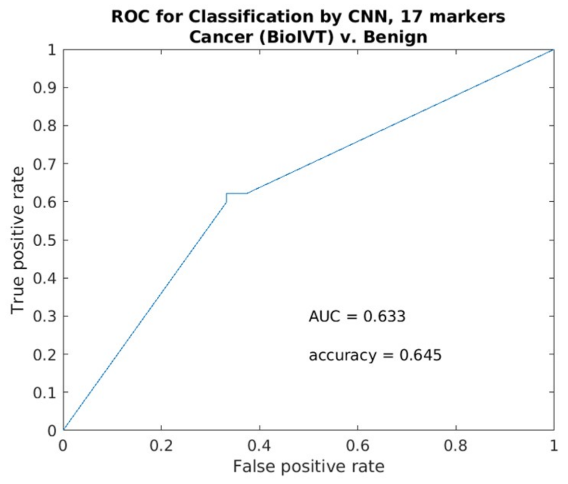


Figure 5.12. ROC curve of Cancer(UA1)/Benign CNN, 17 markers.

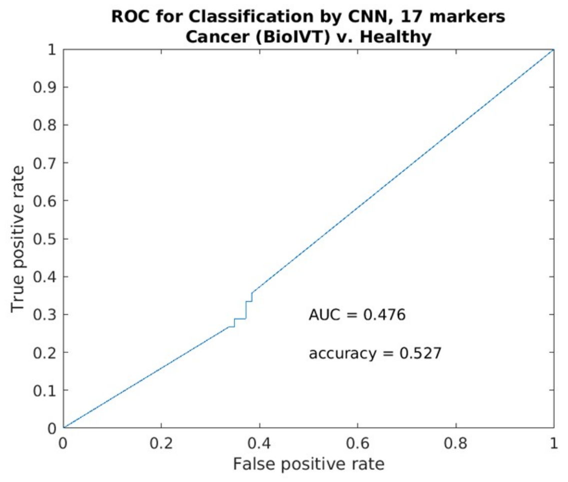


Figure 5.13. ROC curve of Cancer/Healthy CNN, 17 markers.

5.6 Discussion

We present the CNN along with SVM results even though the CNN results are underwhelming. The field of metabolomics and bioinformatics in general is researching on how deep learning can advanced the field beyond the traditional machine learning techniques. However we witness that the small sample size is a major hurdle when applying CNN to metabolomics data for feature extraction or classification. Even a shallow neural network requires a large sample size to avoid overfitting. [105] has demonstrated that network structures could be developed specific to the data types to help overcome the issue of small data.

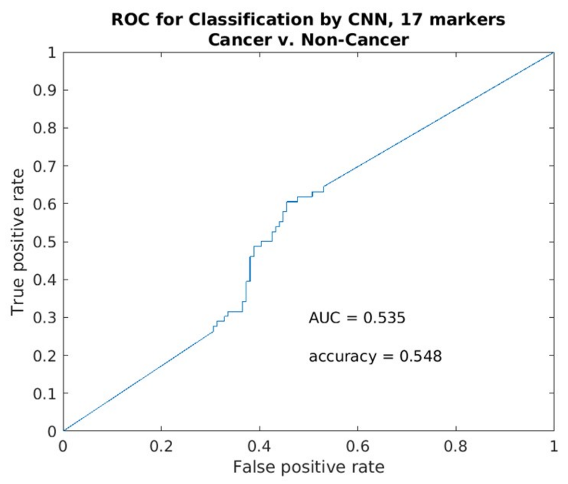


Figure 5.14. ROC curve of Cancer/Non-Cancer CNN, 17 markers.

A second issue is when the omics data are naturally highly noisy. CNN or fully connected networks are successful when the different classes in omics data are well separated, e.g., Figure 5.11, even when sample size is small. On the other hand, highly mixed classes combined with small sample size would require further development of deep learning models for the omics field.

6. CONCLUSION

Building a random ensemble of DNNs has its advantage – the adaptive attacks lose the target. The random ensembles are also able to maintain the same level of accuracy on the clean data as the baseline DNN model. A random ensemble approach is not perfect, as we see the performance against the attacks that add much large amount of perturbations are much lower. Building a robust DNN framework merits more effort. Meanwhile as we have a smaller and more noisy metabolomics data, we see the more classical methods based on biomarker selection and SVM achieves better results than NN. Extending NN model to noisy and small samples will expand its application to a broad set of applications, but also merit more efforts.

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014, pp. 1–10.

[2] X. Yuan, P. He, Q. Zhu, and X. Li, “Adversarial examples: Attacks and defenses for deep learning,” *IEEE transactions on neural networks and learning systems*, pp. 1–20, 2019.

[3] N. Akhtar and A. Mian, “Threat of adversarial attacks on deep learning in computer vision: A survey,” *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[4] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[5] N. Akhtar, A. Mian, N. Kardan, and M. Shah, “Advances in adversarial attacks and defenses in computer vision: A survey,” *IEEE Access*, vol. 9, pp. 155 161–155 196, 2021.

[6] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, *et al.*, “Adversarial attacks and defences competition,” in *The NIPS’17 Competition: Building Intelligent Systems*, Springer, 2018, pp. 195–231.

[7] X. Zhang, H. Chen, and F. Koushanfar, “Tad: Trigger approximation based black-box trojan detection for ai,” *arXiv preprint arXiv:2102.01815*, 2021.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015, pp. 1–12.

[9] A. Rozsa, E. M. Rudd, and T. E. Boult, “Adversarial diversity and hard positive generation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 25–32.

[10] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2017, pp. 1–10.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018, pp. 1–10.

[12] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, “Boosting adversarial attacks with momentum,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9185–9193.

[13] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ACM, 2017, pp. 3–14.

[14] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "Ead: Elastic-net attacks to deep neural networks via adversarial examples," in *Thirty-second AAAI conference on artificial intelligence*, 2018, pp. 1–9.

[15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 372–387.

[16] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 2574–2582.

[17] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[18] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2015, pp. 427–436.

[19] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples models," *arXiv*, vol. 1710.11342, 2017.

[20] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2017, pp. 1–10.

[21] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 15–26.

[22] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[23] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2017, pp. 1–10.

[24] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018, pp. 1–10.

[25] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen, "Dpatch: An adversarial patch attack on object detectors," in *The AAAI's Workshop on Artificial Intelligence Safety (SafeAI 2019)*, 2019, pp. 1–8.

[26] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv*, vol. 1708.06733, 2017.

[27] Y. Li, S. Ma, Y. AAfer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," *Department of Computer Science Technical Reports*, vol. 1781, 2017.

[28] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy*, 2016, pp. 582–597.

[29] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1633–1645, 2020.

[30] Y.-C. Lin, M.-Y. Liu, M. Sun, and J.-B. Huang, "Detecting adversarial attacks on neural network policies with visual foresight," *arXiv*, vol. 1710.00814, 2017.

[31] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," *CCS*, 2017.

[32] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," 2017.

[33] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv*, vol. 1702.06280, 2017.

[34] Z. Gong, W. Wang, and W.-S. Ku, "Adversarial and clean data are not twins," *arXiv*, vol. 1704.04960, 2017.

[35] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *arXiv*, vol. 1703.00410, 2017.

[36] A. N. Bhagoji, D. Cullina, and P. Mittal, "Dimensionality reduction as a defense against evasion attacks on machine learning classifiers," *arXiv*, vol. 1704.02654, 2017.

[37] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," 2017.

[38] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," *arXiv*, vol. 1704.00103, 2017.

[39] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *arXiv*, vol. 1807.03888, 2018.

[40] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bangio, "Binarized neural networks: Training neural networks with weights and activations constrained to +1 or 1," *arXiv*, vol. 1602.02830, 2016.

[41] A. Nguyen, J. Yosinski, and J. Clune, "Attacking binarized neural networks," in *Proceedings of the 6th international conference on learning representations (iclr)*, 2018.

[42] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv*, vol. 1412.6980v9, 2017.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv*, vol. 1502.01852, 2015.

[44] M. Shin, H. Cho, H.-s. Min, and S. Lim, "Neural bootstrapper," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16 596–16 609, 2021.

[45] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," in *ICLR Workshop*, 2015.

[46] G. Paass, "Assessing and improving neural network predictions by the bootstrap algorithm," *Advances in Neural Information Processing Systems*, vol. 5, 1992.

[47] J. Nixon, B. Lakshminarayanan, and D. Tran, "Why are bootstrapped deep ensembles not better?" In *"I Can't Believe It's Not Better!"NeurIPS 2020 workshop*, 2020.

[48] J. Franke and M. H. Neumann, "Bootstrapping neural networks," *Neural computation*, vol. 12, no. 8, pp. 1929–1949, 2000.

[49] J. Lee, Y. Lee, J. Kim, E. Yang, S. J. Hwang, and Y. W. Teh, "Bootstrapping neural processes," *Advances in neural information processing systems (NIPS)*, vol. 33, pp. 6606–6615, 2020.

[50] M. K. Tiwari and C. Chatterjee, "Uncertainty assessment and ensemble flood forecasting using bootstrap based artificial neural networks (banns)," *Journal of Hydrology*, vol. 382, no. 1-4, pp. 20–33, 2010.

[51] E. Zio, “A study of the bootstrap method for estimating the accuracy of artificial neural networks in predicting nuclear transient processes,” *IEEE Transactions on Nuclear Science*, vol. 53, no. 3, pp. 1460–1478, 2006.

[52] J. Ji, Y. Sun, F. Kong, and Q. Miao, “A construction approach to prediction intervals based on bootstrap and deep belief network,” *IEEE Access*, vol. 7, pp. 124 185–124 195, 2019.

[53] H. Du, E. Barut, and F. Jin, “Uncertainty quantification in cnn through the bootstrap of convex neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 12 078–12 085.

[54] E. Nalisnick and P. Smyth, “The amortized bootstrap,” in *ICML 2017 Workshop on Implicit Models*, 2017.

[55] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” *Advances in neural information processing systems*, vol. 30, 2017.

[56] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural network,” in *International conference on machine learning*, 2015, pp. 1613–1622.

[57] U. Simsekli, L. Sagun, and M. Gurbuzbalaban, “A tail-index analysis of stochastic gradient noise in deep neural networks,” in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 5827–5837.

[58] G. Franchi, A. Bursuc, E. Aldea, S. Dubuisson, and I. Bloch, “Tradi: Tracking deep neural network weight distributions,” in *European Conference on Computer Vision*, 2020, pp. 105–121.

[59] M. Mahoney and C. Martin, “Traditional and heavy tailed self regularization in neural network models,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 4284–4293.

[60] C. H. Martin and M. W. Mahoney, “Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning.,” *J. Mach. Learn. Res.*, vol. 22, no. 165, pp. 1–73, 2021.

[61] M. Vladimirova, J. Arbel, and P. Mesejo, “Bayesian neural networks become heavier-tailed with depth,” in *NeurIPS 2018-Thirty-second Conference on Neural Information Processing Systems*, 2018, pp. 1–7.

[62] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[63] B. Efron, “Bootstrap Methods: Another Look at the Jackknife,” *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979. DOI: 10.1214/aos/1176344552. [Online]. Available: https://doi.org/10.1214/aos/1176344552.

[64] B. Efron, “Better bootstrap confidence intervals,” *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 171–185, 1987. DOI: 10.2307/2289144. [Online]. Available: https://doi.org/10.2307/2289144.

[65] M. R. Chernick, *Bootstrap Methods: A Practitioner's Guide*. New York: Wiley, 1999.

[66] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv*, vol. 1409.1556, 2015.

[67] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *arXiv*, vol. 1512.00567v3, 2015.

[68] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *arXiv*, vol. 1512.03385, 2015.

[69] F. J. Massey, “The kolmogorov-smirnov test for goodness of fit,” *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 65–78, 1951. DOI: 10.2307/2280095. [Online]. Available: https://doi.org/10.2307/2280095.

[70] L. N. Vaserstein, “Markov processes over denumerable products of spaces, describing large systems of automata,” *Problems Inform. Transmission*, vol. 5, no. 3, pp. 47–52, 1969.

[71] A. Ramdas, N. Garcia, and M. Cuturi, “On wasserstein two sample testing and related families of nonparametric tests,” 2015. DOI: 10.48550/ARXIV.1509.02237. [Online]. Available: https://arxiv.org/abs/1509.02237.

[72] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, “Barrage of random transforms for adversarially robust defense,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6528–6537.

[73] R. Siegel, D. Naishadham, and A. Jemal, “Cancer statistics,” *CA Cancer J Clin*, vol. 62, pp. 10–29, 2012.

[74] B. M. Reid, J. B. Permuth, and T. Sellers, “Epidemiology of ovarian cancer: A review,” *Cancer Biology Medicine*, vol. 14, pp. 9–32, 2017.

[75] R. A. Smith, D. Manassaram-Baptiste, D. Brooks, M. Doroshenk, S. Fedewa, D. Saslow, O. W. Brawley, and R. Wender, “Cancer screening in the united states, 2015: A review of current american cancer society guidelines and current issues in cancer screening,” *CA Cancer J Clin*, vol. 65, pp. 30–54, 2015.

[76] N. Kozar, K. Kruusmaa, M. Bitenc, R. Argamasilla, A. Adsuar, N. Goswami, D. Arko, and I. Takač, "Metabolomic profiling suggests long chain ceramides and sphingomyelins as a possible diagnostic biomarker of epithelial ovarian cancer," *Clin. Chim. Acta*, vol. 481, pp. 108–114, 2018.

[77] J. T. Henderson, E. M. Webber, and G. F. Sawaya, "Screening for ovarian cancer updated evidence report and systematic review for the us preventive services task force," *JAMA*, vol. 319, pp. 595–606, 2018.

[78] I. A. Yakasai and L. A. Bappa, "Diagnosis and management of adnexal masses in pregnancy," *J Surg Tech Case Report*, vol. 4, pp. 79–85, 2012.

[79] K. Pepin, M. G. del Carmen, A. K. Brown, and D. S. Dizon, "Ca 125 and epithelial ovarian cancer: Role in screening, diagnosis, and surveillance," *Am. J. Hematol. / Oncol.*, vol. 10, pp. 22–29, 2014.

[80] T. S. Hillard, "The impact of mesothelin in the ovarian cancer tumor microenvironment," *Cancers (Basel)*, vol. 10, p. 277, 2018.

[81] S. Wei, H. Li, and B. Zhang, "The diagnostic value of serum he4 and ca-125 and roma index in ovarian cancer," *Biomed Rep*, vol. 5, pp. 41–44, 2016.

[82] N. Husseinzadeh, "Status of tumor markers in epithelial ovarian cancer has there been any progress? a review," *Gynecol. Oncol*, vol. 120, pp. 152–157, 2011.

[83] D. Caserta, M. Benkhalifa, M. Baldi, F. Fiorentino, M. Qumsiyeh, and M. Moscarini, "Genome profiling of ovarian adenocarcinomas using pangenomic bacs microarray comparative genomic hybridization," *Mol. Cytogenet*, vol. 1, p. 10, 2008.

[84] S. Patel and S. Ahmed, "Emerging field of metabolomics: Big promise for cancer biomarker identification and drug discovery," *J Pharm Biomed Anal*, vol. 107, pp. 63–74, 2015.

[85] A. K. Kaushik and R. J. DeBerardinis, "Applications of metabolomics to study cancer metabolism," *Biochim Biophys Acta Rev Cancer*, vol. 1870, pp. 2–14, 2018.

[86] O. Turkoglu, A. Zeb, S. Graham, T. Szyperski, J. B. Szender, K. Odunsi, and R. Bahado-Singh, "Metabolomics of biomarker discovery in ovarian cancer: A systematic review of the current literature," *Metabolomics*, vol. 12, p. 60, 2016.

[87] P. Jasbi, D. Wang, S. L. Cheng, Q. Fei, J. Y. Cui, L. Liu, Y. Wei, D. Raftery, and H. Gu, "Breast cancer detection using targeted plasma metabolomics," *J Chromatogr B Analyt Technol Biomed Life Sci*, vol. 1105, pp. 26–37, 2019.

[88] M. R. Cardoso, J. C. Santos, M. L. Ribeiro, M. C. R. Talarico, L. R. Viana, and S. F. M. Derchain, "A metabolomic approach to predict breast cancer behavior and chemotherapy response," *Int J Mol Sci*, vol. 19, p. 617, 2018.

[89] T. Zhang, X. Wu, M. Yin, L. Fan, H. Zhang, F. Zhao, W. Zhang, C. Ke, G. Zhang, Y. Hou, X. Zhou, G. Lou, and K. Li, "Discrimination between malignant and benign ovarian tumors by plasma metabolomic profiling using ultra performance liquid chromatography/mass spectrometry," *Clin Chim Acta*, vol. 413, pp. 861–868, 2012.

[90] C. Ke, A. Li, Y. Hou, M. Sun, K. Yang, J. Cheng, J. Wang, T. Ge, F. Zhang, Q. Li, J. Li, Y. Wu, G. Lou, and K. Lia, "Metabolic phenotyping for monitoring ovarian cancer patients," *Scientific Reports*, vol. 6:23334, pp. 1–9, 2016.

[91] L. Fan, W. Zhang, M. Yin, T. Zhang, X. Wu, H. Zhang, M. Sun, Z. Li, Y. Hou, X. Zhou, G. Lou, and K. Li, "Identification of metabolic biomarkers to diagnose epithelial ovarian cancer using a uplc/qtof/ms platform," *Acta Oncologica*, vol. 51, pp. 473–479, 2012.

[92] C. Denkert, J. Budczies, T. Kind, W. Weichert, P. Tablack, J. Sehouli, S. Niesporek, D. Könsgen, M. Dietel, and O. Fiehn, "Mass spectrometry-based metabolic profiling reveals different metabolite patterns in invasive ovarian carcinomas and ovarian borderline tumors," *Cancer Research*, vol. 66, pp. 10 795–10 804, 2006.

[93] M. Y. Fong, J. McDunn, and S. S. Kakar, "Identification of metabolites in the normal ovary and their transformation in primary and metastatic ovarian cancer," *PLoS One*, vol. 6, pp. 1–12, 2011.

[94] E. I. Braicu, S. Darb-Esfahani, W. D. Schmitt, K. M. Koistinen, L. Heiskanen, P. Pöhö, J. Budczies, M. Kuhberg, M. Dietel, C. Frezza, C. Denkert, J. Sehouli, and M. Hilvo, "High-grade ovarian serous carcinoma patients exhibit profound alterations in lipid metabolism," *Oncotarget*, vol. 8, pp. 102 912–102 922, 2017.

[95] C. M. Slupsky, H. Steed, T. H. Wells, K. Dabbs, A. Schepansky, V. Capstick, W. Faught, and M. B. Sawyer, "Urine metabolite analysis offers potential early diagnosis of ovarian and breast cancers," *Clin Cancer Res*, vol. 16, pp. 5835–5841, 2010.

[96] M. Kyriakides, N. Rama, J. Sidhu, H. Gabra, H. C. Keun, and M. El-Bahrawy, "Metabonomic analysis of ovarian tumour cyst fluid by proton nuclear magnetic resonance spectroscopy," *Oncotarget*, vol. 7, pp. 7216–7226, 2016.

[97] M. F. Buas, H. Gu, D. Djukovic, J. Zhu, C. W. Drescher, N. Urban, D. Raftery, and C. I. Li, "Identification of novel candidate plasma metabolite biomarkers for distinguishing serous ovarian carcinoma and benign serous ovarian tumors," *Gynecol. Oncol*, vol. 140, pp. 138–144, 2016.

[98] J. Zhu, D. Djukovic, L. Deng, H. Gu, F. Himmati, E. G. Chiorean, and D. Raftery, "Colorectal cancer detection using targeted serum metabolic profiling," *J. Proteome Res*, vol. 13, pp. 4120–4130, 2014.

[99] P. A. Carroll, D. Diolaiti, L. McFerrin, H. Gu, D. Djukovic, J. Du, P. F. Cheng, S. Anderson, M. Ulrich, J. B. Hurley, D. Raftery, D. E. Ayer, and R. N. Eisenman, "Deregulated myc requires mondoa/mlx for metabolic reprogramming and tumorigenesis," *Cancer Cell*, vol. 27, pp. 271–285, 2015.

[100] H. Gu, P. Zhang, J. Zhu, and D. Raftery, "Globally optimized targeted mass spectrometry: Reliable metabolomics analysis with broad coverage," *Anal. Chem*, vol. 87, pp. 12 355–12 362, 2015.

[101] H. Gu, P. A. Carroll, J. Du, J. Zhu, F. C. Neto, R. N. Eisenman, and D. Raftery, "Quantitative method to investigate the balance between metabolism and proteome biomass: Starting from glycine," *Angew. Chemie Int. Ed Engl*, vol. 55, pp. 15 646–15 651, 2016.

[102] R. Li, S. A. Grimm, D. Mav, H. Gu, D. Djukovic, R. Shah, B. A. Merrick, D. Raftery, and P. A. Wade, "Transcriptome and dna methylome analysis in a mouse model of diet-induced obesity predicts increased risk of colorectal cancer," *Cell Reports*, vol. 22, pp. 624–637, 2018.

[103] M. F. Buas, H. Gu, D. Djukovic, J. Zhu, L. Onstad, B. J. Reid, D. Raftery, and T. L. Vaughan, "Candidate serum metabolite biomarkers for differentiating gastroesophageal reflux disease, barrett's esophagus, and high-grade dysplasia/esophageal adenocarcinoma," *Metabolomics*, vol. 13, p. 23, 2017.

[104] J. Chong, O. Soufan, C. Li, I. Caraus, S. Li, G. Bourque, D. S. Wishart, and J. Xia, "Metaboanalyst 4.0: Towards more transparent and integrative metabolomics analysis," *Nucleic Acids Res*, vol. 46, W486–W494, 2018.

[105] R. N. D'souza, P.-Y. Huang, and F.-C. Yeh, "Structural analysis and optimization of convolutional neural networks with a small sample size," *Scientific Reports*, vol. 10, 2020.