ON THE ROBUSTNESS OF QUANTIZED CONVOLUTIONAL NEURAL NETWORKS

by

Jack Langille

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

at

Dalhousie University
Halifax, Nova Scotia
August 2024

Table of Contents

List of Tables

List of Figures

Abstract

This thesis studies the performance and robustness of post-training INT8 quantized convolutional neural networks under various perturbation regimes. Perturbations include additive white Gaussian noise (AWGN), spatially correlated Brownian noise, and structured vertical and horizontal occlusions. Three state-of-the-art models, including VGG-16, ResNet-18, and SqueezeNet1_1, are examined. Performance metrics include top-1 accuracy, top-5 accuracy, and F1 score. We also employ Kullback-Leibler (KL) divergence to measure differences in model confidences in their output class probabilities. We depart from traditional benchmark datasets and instead study fine-grained visual classification to a) better model real-world image classification tasks where specificity of sub-classes derived from a parent class is favored over generality and b) better stress the reduced precision model. This research aims to identify points of instability or ill-conditioning in the quantized model relative to its full-precision version to provide experimental bounds on quantization for deployment in scenarios where random perturbation may be present, as is common in computer vision systems owing to thermal noise, sensor faults, and environmental conditions. It was found that across all three models and under each perturbation scheme, the relative error between the quantized and full-precision model was consistently low, with the maximum error being in VGG-16 under Brownian noise with a top-1 accuracy drop of 1.62% in the quantized model. We also find that KL divergence was on the same order of magnitude as the unperturbed tests across all perturbation regimes except Brownian noise, where maximum divergences ranged from 1.6631 (VGG-16) to 2.3271 (SqueezeNet1_1). While secondary to quantization-induced errors, it was also observed that, in general, models were most sensitive to vertical occlusions, with accuracy degrading to sub-50% at the lowest level of perturbation.

List of Abbreviations and Symbols Used

ASIC Application-specific integrated circuit.

AWGN Additive white Gaussian noise.

CIFAR Canadian Institute For Advanced Research.

CNN Convolutional neural network.

FFT Fast Fourier transform.

FGVC Fine-grained visual classification.

FGVC-A Fine-grained visual classification-aircraft.

FLOPS Floating-point operations.

FP32 32-bit floating-point.

FPGA Field-programmable-gate arrays.

GFLOP Giga floating point operations.

IFFT Inverse fast Fourier transform.

INT8 8-bit integer.

KDE Kernel density estimation.

KL Kullback-Leibler.

MAC Multiply-accumulate.

MNIST Modified National Institute of Standards and Technology database.

PTQ Post training quantization.

QAT	Quantization aware training.
ReLU	Rectified-linear unit.
SOTA	State-of-the-art.

Acknowledgements

I would like to thank my supervisor, Dr. Issam Hammad, for his guidance throughout this project. His expertise in machine learning was invaluable for its completion. I also thank Dr. Guy Kember for his input and wisdom in experiment design. Without their continued support, this research would not have been possible. Additionally, I extend my gratitude to Dr. Kamal El-Sankary for being a member of my committee and assisting in the initial design of this project.

Chapter 1

Introduction

1.1 Motivation

Convolutional neural networks (CNNs) have emerged as an effective means of modeling relationships in spatially expressive data, introducing a new domain of computer vision tasks ranging from image classification and segmentation to object detection and video processing. The concept of the convolutional layer was introduced in the 1989 paper *Handwritten digit recognition with a back-propagation network* [7]. This concept was refined in 1998 with the introduction of LeNet-5, the first formalized CNN [20]. Since LeNet-5, CNNs have become ubiquitous in the machine learning ecosystem. Today, popular CNN model architectures include AlexNet [19], VGGNet [39], and ResNet [15]. While no doubt impressive in classification accuracy and generalization ability, these models suffer in their increasing parameter counts - as capability increases, so does the model's size. VGGNet is predicated on this concept, as they sought to show that model performance scales with model size [39]. When discussing size in neural networks, we are referring to the parameter count. In CNNs, parameters are predominately comprised of filter weights and activations, both typically stored as 32-bit floating-point numbers (FP32). Weights are the model's learnable parameters, and activations are the outputs of input data passing through non-linear functions like a rectified linear unit (ReLU). The convolution operation repeatedly computes a filter's dot product with a corresponding receptive field as it slides over the entire input space, as shown in Figure 1.1. Each filter embeds a specific spatial pattern, such as a horizontal edge, and is learned during training.

In hardware, this process is performed by multiply-accumulate (MAC) blocks, which are resource-intensive compared to other standard processor operations. Further, using FP32 introduces complexities to the operation, as floating-point operations must keep track of the exponent, mantissa, and special numbers like *NaN* and *infinity*. The increased bit-width of a floating-point number requires more memory and

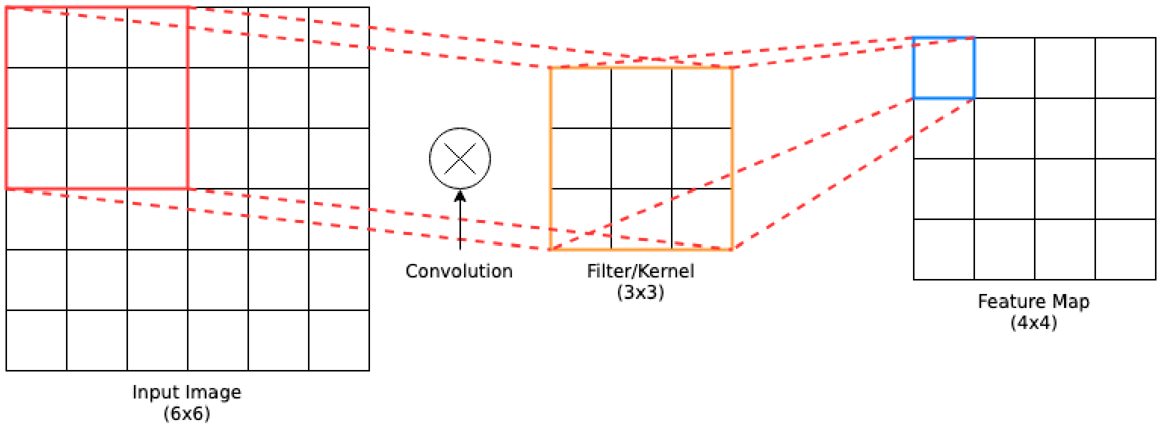


Figure 1.1: Visual depiction of a convolution operation. Weights are the elements of the filter/kernel, and activations are the result of passing the feature map's elements through an activation function.

cache, yielding slower data movement per memory access. This results in slower forward passes, higher memory consumption, and increased disk space for storage. This also has other downstream effects, such as increased heat emission and power draw [28] [40].

Thus, there is motivation to reduce such networks' complexity and size while maintaining original model performance. To this end, there are several approaches, such as pruning [2], low-rank tensor approximations [31], approximate multipliers [14] [23] and quantization [28]. This thesis will examine quantization, as it has emerged as a popular approach to the problem of network optimization [40][28][41]. Quantization differs from pruning and tensor approximations in that it does not seek to explicitly remove or drop parameters but rather reduce the precision of parameters. To illustrate this, consider VGG-16, which has 138 million parameters [39], consuming roughly 530 megabytes of memory and disk space. If, through some quantization scheme, we reduced the bit-width of each parameter to that of an 8-bit integer (INT8), our model space consumption would decrease by a factor of 4 to 130 megabytes and significantly decrease inference latency. For instance, [11] observed an inference latency decrease from 131.8 ms to 3.5 ms with ResNet-50 by implementing 8-bit quantization! This is particularly attractive in the context of embedded systems as hardware such as field-programmable-gate arrays (FPGAs) and application-specific integrated circuits (ASICs) can be designed to use any bit-width numerical representation [40] and are often expected to have near real-time processing and are resource constrained [12] [14].

As noted, a crucial aspect when reducing a model's complexity is defining performance criteria and understanding how it changes. Much of the current machine learning discourse concerns raw accuracy or other metrics such as the F1 score. While these metrics are indeed essential and grant an immediate understanding of the proportion of correct predictions, they can be misleading alone. More precisely, such metrics are binary; they consider the case of a true or false prediction. The issue here is that they offer no insight into *how confident* a model is in its prediction. This notion is particularly important in quantized/base model pairs, as to de-risk the lack of precision introduced by quantization, we must fully understand how a given model's *robustness* is changing. To this end, we pair traditional metrics of accuracy and F1 score with an examination of the output class probabilities in a given full precision/quantized model pair using Kullback-Leibler (KL) divergence.

To the best of our knowledge, studies relating to quantization have solely evaluated their quantization strategies using datasets in which the test set is no different than the training set. More precisely, the data used to test quantized models is unperturbed relative to the training set. This reveals a significant gap, as real-world applications, ranging from autonomous systems to medical imaging, frequently encounter unexpected perturbations and various forms of noise. Moreover, adversarial attacks designed to fool CNNs by deliberately perturbing the input space have become an increasing area of study [37]. The absence of rigorous testing under these conditions poses a substantial risk. A lack of insight into the robustness of quantized networks against such perturbations could result in unforeseen failures in model deployment owing both to innocuous random environmental noise and deliberate adversarial attacks. To be precise, we consider robustness to be the performance of a model when faced with external perturbations to its input space, e.g. noise. This is in contrast to *resiliency*, which is concerned with the model's architectural features itself, i.e. if model features change or are lost entirely, how does the model's performance degrade?

Furthermore, we depart from standard benchmarks such as ImageNet [8], Modified National Institute of Standards and Technology database (MNIST) [21], and

Canadian Institute For Advanced Research (CIFAR) [18] and instead study fine-grained visual classification (FGVC), using the fine-grained visual classification aircraft (FGVC-A) dataset [26]. The goal of FGVC is to classify subcategories of a broader category, in this case, families of aircraft. The rationale for this is several-fold. Firstly, the differences between classes in FGVC are comparatively subtle[1]. It is expected that as these subtleties are perturbed, we are better stressing the reduced precision model as its expressive power, or capacity for spatial embedding, is limited.

Conversely, if degradation is not observed, it suggests that lower precision models are *as* robust as their full precision counterparts even under unseen perturbations, further proving their viability for real-world deployment and further supporting the notion that contemporary models have inherent redundancies beyond that of offering robustness. The MNIST dataset was also not examined due to its simplistic nature - low-resolution monochromatic images - and subsequent lack of relevance in real-world scenarios. Further, it is not expected that quantization would challenge an MNIST-trained model as each of the models studied has achieved near-perfect or perfect accuracy on MNIST. Lastly, we contend that studying FGVC is more relevant to practical computer vision implementations, where a model is trained to classify subcategories of a parent category - rather than generally classifying a wide range of uncorrelated classes - as is often the case in medical imaging, autonomous vehicles, and surveillance systems. We hope this study will serve as an objective contribution to understanding how quantized networks trained for targeted FGVC tasks behave against unseen perturbations and in doing so, characterize their viability in real-world applications.

1.2 Related Work

As noted earlier, neural network quantization has garnered popularity with the growing demand for edge computing and efficient hardware utilization. [28] provides an essential reference for the mechanics of neural network quantization and will be used as the primary source when implementing our quantizers. Specifically, they introduce and discuss the design parameters when implementing quantization, including the bit

[1]A helpful illustration of this is considering the visual differences between a Boeing 737 and a Boeing 777, versus the differences between a hummingbird and a school bus, all of which are classes in [26] and [8] respectively.

resolution, quantization range, scale factor, zero point, and level of granularity (per tensor and channel). Additionally, they define two main classes of quantization algorithms: quantization aware training (QAT) and post-training quantization (PTQ). [28] also provides experimental results for their quantizers under various configurations for common model architectures, which is crucial, as it sets a ground truth for what we should expect in terms of performance under the same quantization scheme. The primary takeaway from this work is the choice to study PTQ. This is because, in most cases, PTQ is sufficient for 8-bit quantization while maintaining close to FP32 accuracy [28]. Moreover, PTQ is simpler than QAT as it does not require a complete re-training step but rather a simple calibration set without needing a labeled dataset. While it is true that QAT generally permits lower bit resolutions (as low as 1-bit), we feel it is not relevant as to yield the benefits of sub-8-bit resolutions, specialized hardware is required [40]. That is to say, 8-bit quantization is more prevalent in its adoption in real-world deployment and, thus, of more relevance to this research.

While still based on the core fundamentals outlined in [28], novel quantization schemes based on different optimizations or target criteria have been a very popular area of research in the past decade. [24] designed an optimal post-training quantizer scheme based on optimizing the signal-to-quantization-noise ratio. They used statistics about the underlying distribution of weights and activations to determine the optimal bit resolution and step size - interestingly, as will be discussed later, this is how frameworks like PyTorch implement quantization [32]. This approach was tested with an AlexNet-like model on the CIFAR-10 dataset, in which they achieved 4-bit weight and activation quantization with an error rate of 8.30% [24]. [42] implemented a filter clustering approach to create quantized "codebooks" for weights and activations, permitting high compression rates. On VGG-16, this method achieved a 30.5$\times$ compression with an error rate of just 0.2% [42]. Other approaches include ternary weight networks [22] and binarized neural networks [5] [35]. While these are certainly advances in low-precision machine learning, there is a commonality of only measuring accuracy and, further, considering standard benchmark datasets like ImageNet, CIFAR, and MNIST. That is, there is ample evidence to suggest that the current landscape of quantization has not yet considered a) targeted FGVC tasks, b) changes in confidence, and c) quantization robustness under perturbation.

In contrast, several studies examine the performance of *full-precision* networks under perturbation. [9] studied the impacts of Gaussian blur, additive white Gaussian noise (AWGN), JPEG compression, and contrast manipulation on common state-of-the-art (SOTA) networks including CaffeNet, VGG-16, VGG-CNN-S, and GoogleNet, pre-trained on ImageNet. Their experiments revealed several key insights that are complimentary to this work. Firstly, they found that in terms of accuracy, each model was most susceptible to blur. As they note, this is fairly intuitive as the feature space of an CNN is a collection of textures and edges, and any distortions therein will subsequently degrade model performance. Another interesting result is that each model degrades at similar rates with increasing perturbation, with VGG-16 and GoogleNet exhibiting marginally higher robustness, most likely owing to their relative depth. [10] also studied model performance under perturbation, but rather than studying known perturbation sources like blur, AWGN, or compression, they explore three mathematically defined noise regimes: random noise, semi-random noise, and worst-case (adversarial) perturbations. They define robustness as the minimal perturbation required to change a model's output, with the random noise regime involving perturbations in random directions within the input space and the semi-random noise regime introducing an adversarial component by selecting sub-spaces of varying dimensions of the input. They relate the robustness of a model to its decision boundary curvature and show that when this boundary has small curvature, models are robust to random noise in high dimensional classification problems [10]. However, for semi-random noise with an adversarial component, each model studied was deemed not robust and vulnerable.

As noted in the previous section, we are interested in extending our measure of performance beyond measures of accuracy to characterize confidence and robustness. [30] investigates this with the goal of better understanding the tension between model complexity and generalization capacity. They note that typically, large networks generalize better than smaller counterparts, an observation that contradicts traditional notions of function complexity. This question is relevant to this research as we are balancing these two extremes; we are reducing the parameter precision and, thereby, information storage capacity, but we are not reducing the overall parameter count of the network. [30] develops three noise regimes, each of which is defined with respect to the training data manifold: a) a random ellipse, unlikely to pass near actual data, b) an ellipse passing through three training points, and c) an ellipse passing through three training points of the same class. They then analyze the model's sensitivity by computing the average Frobenius norm for the batch Jacobian matrices of each model's output neurons with respect to the input space, i.e., answering the question: *how does the model's output confidence change as the input space is perturbed?*. From this, several results emerge. Firstly, they observe that generalization is strongly correlated with sensitivity by observing that factors conducive to generalization, such as data augmentation, and ReLU activation functions, yield more robust models [30]. Secondly, they find that in all cases, model robustness degrades as input data departs from the training data manifold. The limitation of this approach is that it relies on the assumptions that perturbations are small such that the output activation function can be well-approximated by its first order Taylor expansion - large perturbations would require a more accurate approximation using higher order terms. Secondly, this method assumes perturbations are Gaussian in nature, for mathematical convenience. Given we wish to extend our study of noise beyond uncorrelated random perturbations to highly structured noise, we will not be employing this measure of robustness.

In summary, these studies illustrate a rich but fragmented area of research. Plenty of work has been done on quantization methods and schemes, all yielding impressive results. Others have studied network performance under perturbation, while others have studied and proposed sensitivity and robustness metrics. As quantized networks continue to garner attention in embedded devices and edge applications, so will the relevance of this gap. This thesis aims to fill this gap by integrating each of these topics into a coherent characterization of quantized network performance under various perturbation regimes.

Chapter 2

Theory and Background

2.1 Quantization

Quantization is the non-linear mapping of a domain of continuous amplitude inputs onto a finite set of output levels. In neural network quantization, we are concerned with the mapping of the continuous set of possible FP32 numbers to a finite subset of INT8 numbers. In accordance with typical approaches to network quantization, we are quantizing the weights and outputs (activations) of each convolutional filter [28] [40] [41]. To realize the gains offered by INT8 arithmetic, specialized hardware for INT8 MAC operations must be used. Namely, this involves quantizing and converting the weights and inputs to INT8, performing MAC operations, and quantizing and converting the output, or activations, to INT8. However, frameworks such as PyTorch typically use FP32 for parameter representation. We reconcile this gap by performing "fake quantization", in which we map each FP32 parameter to its nearest quantization level on an INT8 quantization grid. Thus, our quantized model is not a true INT8 model, and we will not experience inference time reductions, energy savings, or hardware optimizations. For the purposes of this research, fake quantization is entirely appropriate as we are solely concerned with numerical and statistical characteristics of a quantized network. In theory, these two schemes should yield the exact same results in terms of model performance, as the size of set of possible weights and activations is the same in both real and fake quantization. To achieve true INT8 quantization, one would need to perform an additional conversion step, targeted for specific hardware.

Figure 2.1 illustrates the process of fake quantization, where 100 FP32 parameters, sampled from a standard normal distribution, are mapped to a symmetric 4-bit quantization grid. We use 4 bits for this example to clearly visualize the quantization process, as an INT8 grid's 256 levels would be too dense to effectively illustrate.

In the plot, we observe that the original FP32 values, which range approximately from -2 to +2, are now mapped to discrete levels between -8 and +7 in the quantized grid. This new range is determined by the 4-bit resolution, providing 16 possible quantization levels. Because the grid is symmetric about 0, the range extends from -2^{b-1} to $2^{b-1}-1$. In this case, the scale is set to 0.1, meaning each quantization level represents a 0.1 increment in the original floating-point values.

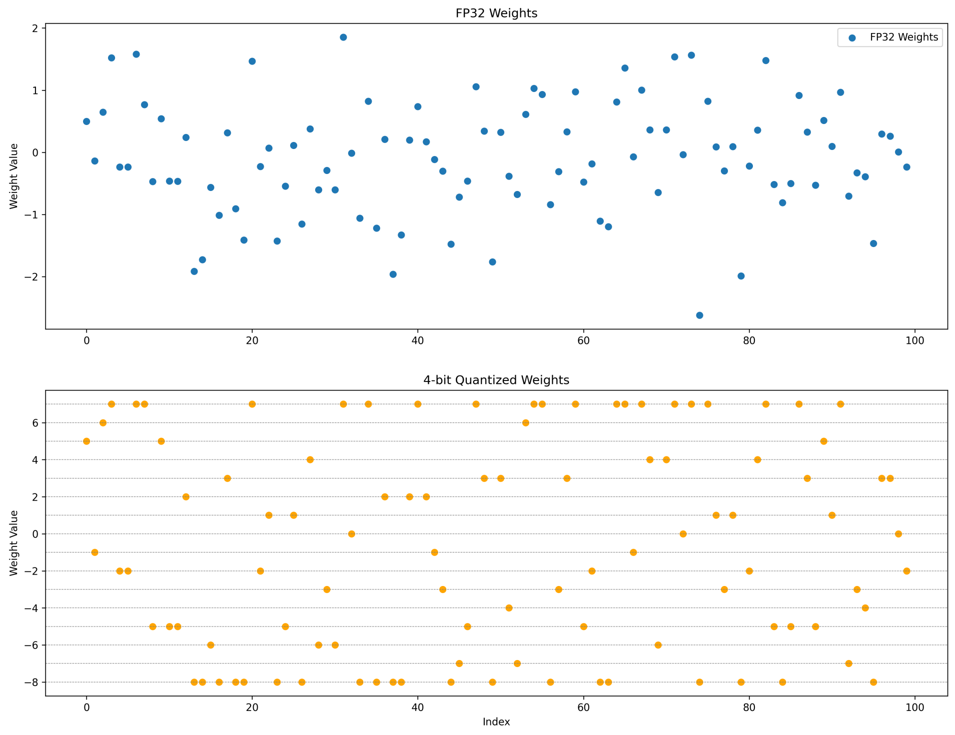


Figure 2.1: FP32 weights before (top) and after (bottom) 4-bit fake quantization.

While it is possible to train a quantized network entirely from scratch, other works have shown that starting with a pre-trained network is more effective in preserving model performance [40] [41] [28]. To this end there are two primary approaches: quantization-aware training (QAT) and post-training quantization (PTQ). QAT is the process of quantizing the desired parameters and performing a re-training step in which the quantized parameters are optimized such that quantization errors and biases are minimized [28]. QAT is shown to yield negligible accuracy loss and high compression ratios, allowing for as low as 1 bit quantization, but comes at the cost of additional training time and data [40] [28]. In contrast, PTQ allows for much faster quantization, as it does not require a full re-training step, but rather a small calibration step using unlabeled data to optimize quantization parameters. PTQ is favourable as it still yields relatively low accuracy loss, and has been shown to enable quantization down to as low as 4 bits [40].

This thesis will exclusively consider uniform PTQ. That is, quantization on a pre-trained model where the step-size between each quantization level is uniform. This is in contrast to dynamic quantization which seeks to find an optimal arrangement of quantization levels to minimize an error function such as mean squared error, given a set number of bits. While dynamic quantization typically yields less error , such approaches are comparatively niche, with little support in modern machine learning frameworks. The interested reader is encouraged to look at the seminal work on dynamic quantization, the Lloyd-Max quantization algorithm [25][27].

To perform PTQ one must first define several quantization parameters: the bit-width b, the step-size or scale factor, s, and the zero-point z [28]. The bit-width defines the number of possible levels in the quantization grid. The scale factor sets the step-size, or the difference between each level, and the zero-point is an integer chosen such that actual zero is quantized without error, which is important to ensure activation functions like ReLU do not introduce additional quantization error [28]. Then depending on if one is performing symmetric or affine quantization, parameters are mapped to the quantization grid depending on the symmetry of the scheme. For the unsymmetric (affine) case we have:

$$x_{\text{INT8}} = \text{clamp}\left(\left\lfloor \frac{x_{\text{FP32}}}{s} \right\rceil + z, 0, 2^b - 1\right), \tag{2.1}$$

For the symmetric about z case:

$$x_{\text{INT8}} = \text{clamp}\left(\left\lfloor \frac{x_{\text{FP32}}}{s} \right\rceil + z, -2^{b-1}, 2^{b-1} - 1\right), \tag{2.2}$$

Where $\lfloor\cdot\rceil$ is the round-to-nearest integer operator and the clamping function is defined as:

$$\text{clamp}(x; a, c) = \begin{cases} a & \text{if } x < a, \\ x & \text{if } a \leq x \leq c, \\ c & \text{if } x > c. \end{cases} \tag{2.3}$$

Where a and c denote the bounds of the integer grid. Our quantization range is bounded by $q_{\min}$ and $q_{\max}$, which is defined depending on if the quantization is symmetric or asymmetric. For affine quantization, the quantization range is defined as:

$$q_{\min} = -sz \tag{2.4}$$

$$q_{\max} = s(2^b - 1 - z) \tag{2.5}$$

For the symmetric case, z is constrained to 0, and the range is bounded by:

$$q_{\min} = -s(2^{b-1}) \tag{2.6}$$

$$q_{\max} = s(2^{b-1} - 1) \tag{2.7}$$

The choice between symmetric or affine quantization is entirely dependent on the underlying distribution of the parameter of interest. As noted in [28], ReLU activations are positively skewed, favouring an affine scheme, whereas weights are approximately symmetric around 0, favouring a symmetric scheme. We demonstrate this by examining kernel density estimation (KDE) plots of a convolutional layer from ResNet-18, as shown in 2.2 in which such characteristics are evident.

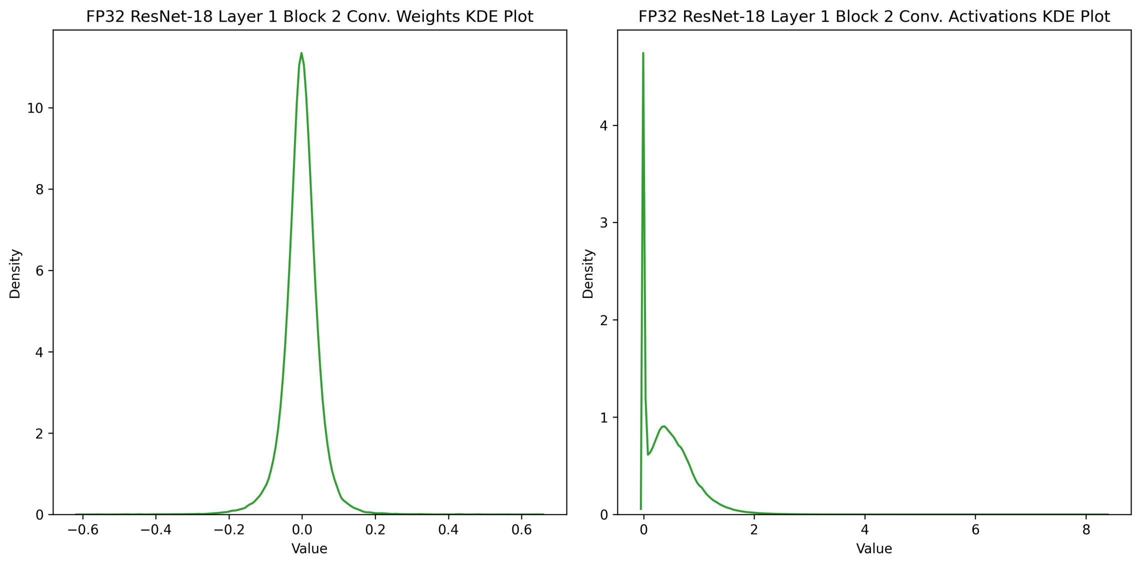


Figure 2.2: KDE plots of full precision ResNet-18 weights and activations.

Given the non-linear many-to-few nature of quantization, it is considered a *lossy* compression scheme, and has inherent error that increases as the bit resolution is decreased. This error owes largely to two sources - clipping and rounding error, combined, referred to as *quantization error*. Clipping error is induced when data laying outside of the dynamic range of the quantizer is rounded to the nearest minimum or maximum bound. We can minimize clipping error by increasing the range of the quantization grid with a larger scale factor s. However this comes at the cost of increasing the second type of error, rounding error, as we are increasing the distance between each quantization level. Thus an optimal quantizer would balance this trade off to minimize some error metric. To determine this optimal set of parameters, observation based techniques are typically employed such that quantization error is minimized.

In practice, frameworks are typically used to interact with and quantize existing model architectures. For this research we use PyTorch [32] and its quantization module paired with a third party wrapper library, EasyQuant [36]. At the core of this quantization module is the above mentioned observer. Observers seek to determine optimal quantization parameters by collecting statistics about weights and activations during the PTQ calibration step, which, again, is favourable compared to QAT as it does not require a full set of labeled data - a random tensor with appropriate dimensions is entirely sufficient. PyTorch has several observer modules that employ various algorithms. For this research we employ a min-max observer for weights and a histogram observer for activations. The min-max observer is a comparatively simple algorithm that tracks the minimum and maximum values of the filter weights on a per-channel basis [28]. These min and max values are then used to determine each filter's scale factor and zero point using the formulae shown above. A problem with this algorithm is that it is sensitive to outliers, as such outliers will skew the quantization range. However, this approach is suitable for weights as their values are typically zero or near zero with comparatively low variance, and moreover, weight quantization has been shown to induce less degradation in terms of accuracy, compared to activations [28]. For activations we employ a histogram observer which operates by recording frequencies of activation tensor values during a calibration step. Following this, a histogram is constructed from which the min and max can be set such that outliers are ignored, e.g., by clamping the range to only cover a certain percentile of data. From here, like the min-max observer, quantization parameters can be computed. For activations, this operation is performed on a per-layer basis and for weights, a per filter basis, as per the recommendations outlined in [28]. We can illustrate the characteristics of each scheme further by examining kernel density estimation (KDE) plots of the weigths and activations post quantization, as shown in 2.3. Immediately of note is the higher variance in the quantized plots, this makes sense and helps confirm the validity of our quantization process as we would expect that given the reduced resolution of the quantized values, there would be higher variance or "spread" in their distribution. We also can observe the two distinct peaks at each tail, corresponding to -128 and 127, the limits of 8-bit quantization as $-2^{8-1} = -128$ and $2^{8-1} - 1 = 127$. These peaks are expected as they are the limits of our quantization range, thus any value falling outside of this range will be clamped to these two boundaries. To illustrate these characteristics further, the code snippet below in Listing 1 performs a simple 8-bit symmetric quantization on randomly generated FP32 data and computes their respective variances. As expected, the INT8 data has significantly higher variance.

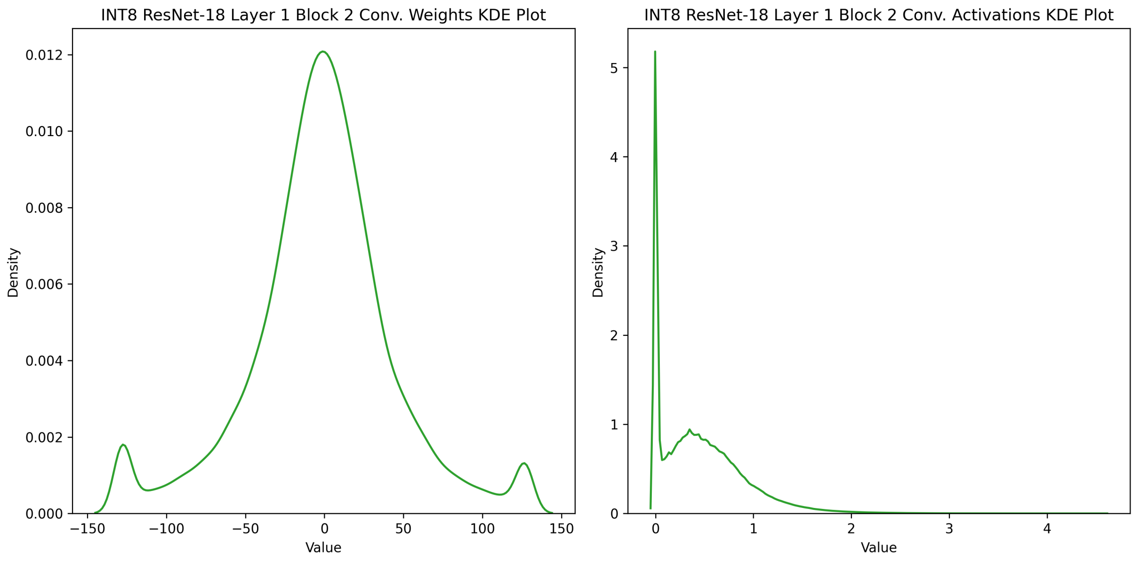


Figure 2.3: KDE plots of quantized ResNet-18 weights and activations.

Listing 1 Python code demonstrating INT8 quantization and subsequent variances.

```python
import numpy as np
np.random.seed(42)
fp32_weights = np.random.randn(10)
s = 0.01     # Scale
z = 0        # Zero point
b = 8        # Bit resolution
q_min = -(2**(b-1))      # Minimum quantization value
q_max = (2**(b-1)) - 1  # Maximum quantization value
int8_weights = np.clip(np.round(fp32_weights / s)
    + z, q_min, q_max)
variance_fp32 = np.var(fp32_weights)
variance_int8 = np.var(int8_weights)
print(fp32_weights)
print(int8_weights)
print(f"Variance of FP32 weights: {variance_fp32}")
print(f"Variance of INT8 weights: {variance_int8}")
```

```
Output:
[ 0.49671415 -0.1382643   0.64768854  1.52302986 -0.23415337
-0.23413696  1.57921282  0.76743473 -0.46947439  0.54256004]
[ 50. -14.  65. 127. -23. -23. 127.  77. -47.  54.]
Variance of FP32 weights: 0.4704669452131567
Variance of INT8 weights: 3584.6099999999997
```

2.2 Input Perturbations

We now turn our discussion to the topic of perturbations. Recall our goal is to observe the relative degradation, if any, induced in the quantized model by perturbing the input space. In image classification tasks the input space is the flattened vector of an image's pixel intensities, $\mathbf{x}$. That is:

$$\mathbf{x} \in \mathbb{R}^{H \times W \times C} \tag{2.8}$$

where H is the height, W is the width, and C is the number of channels, e.g. a red, green and blue image has 3 channels. In this research we examine four forms of perturbation, additive white Gaussian noise (AWGN), Brownian noise, and vertical and horizontal structured occlusions. As will be discussed, each of these have relevance in real-world scenarios and were chosen for varying expected impacts on model performance.

AWGN is a common approach used in noise modelling as it approximates many real world phenomena. For example, modelling thermal noise and in the limiting behaviour of other noises such as photon counting and film grain artifacts [13] [3]. In AWGN, each pixel's noise term is sampled independently and identically from the same Gaussian distribution, that is, there is no correlation between pixels. Consequently this yields an image that appears grainy, similar to film grain or tv static, as shown in Figure 2.4. We induce AWGN noise in our input image tensor $\mathbf{X}$ as follows. Given the univariate Gaussian density function with mean $\mu = 0$ and variance σ^2:

$$\mathcal{N}(0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x)^2}{2\sigma^2}} \tag{2.9}$$

We sample noise on a per-pixel basis, where pixels are indexed by i, j, and k denoting row, column and channel, respectively, yielding our C channel additive noise mask $n_g(i, j, k)$:

$$n_g(i, j, k) \sim \mathcal{N}(0, \sigma^2) \tag{2.10}$$

We then generate our perturbed input through simple addition of the noise max to the original input:

$$X'(i, j, k) = X(i, j, k) + n_g(i, j, k) \tag{2.11}$$

Note that we cannot directly use 2.11 as is, as there is a non-zero chance the resulting pixel will be non-negative. To ensure this does not occur, we clamp the noise distribution, 2.9, to be between $\pm 3\sigma$ [3]. In this scheme, we control the "intensity" of the noise with the standard deviation σ, that is, increasing standard deviation increases the distortion in the image since higher standard deviation means the sampled noise values have a higher chance of larger deviation from the zero-valued mean.

Red noise, also known as Brownian noise, differs from AWGN in that it is *not* independent and identically distributed, but rather, spatially correlated. Moreover, unlike the flat power spectral density of AWGN, Brownian noise has a power spectral density proportional to the inverse square of frequency, $\frac{1}{f^2}$. Consequently, lower frequencies dominate the noise spectrum. Structurally, pixels are correlated in the sense that smaller frequencies are amplified, where as higher frequencies are attenuated, yielding long range correlations in the spatial domain and smooth variations. In effect, Brownian noise adds blotches and blur-like artifacts to an image, as illustrated in Figure 2.4. Brownian noise is particularly relevant in vision systems for modelling natural phenomena like underwater distortions, blur, clouds, as well as in medical imaging [34] [4].

To generate Brownian noise, we start with white noise, $n_g(i,j,k)$, sampled from 2.9. We then apply the fast Fourier transform (FFT) to this noise on a per channel basis, that is for each channel, compute the frequency component, $W(u,v,z)$:

$$W(u,v,z) = \text{FFT}[n_g(i,j,k)] \tag{2.12}$$

We then scale this frequency component according to the $\frac{1}{f^2}$ rule:

$$B(u,v,z) = \frac{W(u,v,z)}{(u^2+v^2+z^2)} \tag{2.13}$$

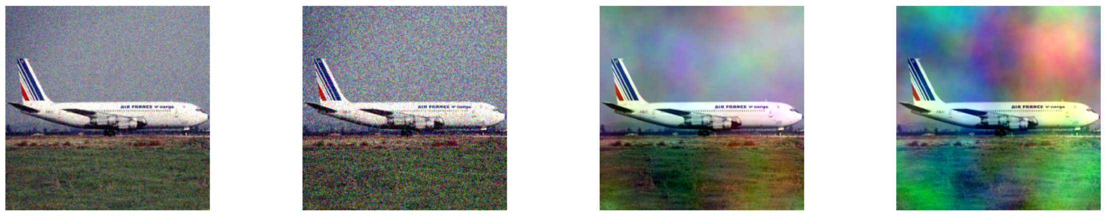
Finally, we apply the inverse FFT (IFFT)to convert back to the spatial domain, and add the resulting noise to the image:

$$b(i,j,k) = \text{IFFT}[B(u,v,z)] \tag{2.14}$$

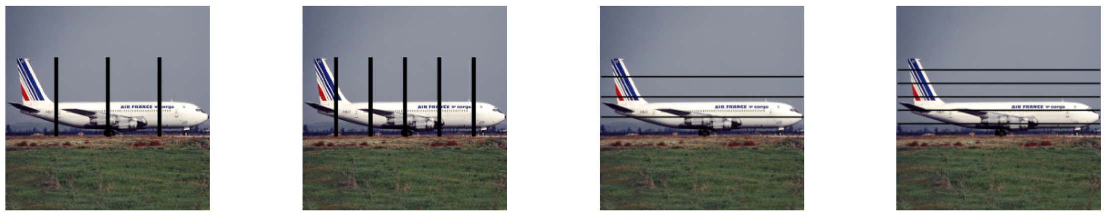
$$X'(i,j,k) = X(i,j,k) + b(i,j,k) \tag{2.15}$$

Similar to AWGN, the intensity factor in Brownian noise is the standard deviation, σ. However, as will be shown when we discuss the experimental procedure, Brownian noise needs a comparatively larger values to yield noticeable distortions and subsequent impacts to model performance. This is because in AWGN, σ directly controls the noise of each pixel where a small σ results in small changes, but as these changes are uniformly distributed across all pixels, the overall perturbation to the image is significant. That is, AWGN does not account for spatial structure in the image, leading to a high-frequency noise pattern that noticeably alters the image even at small values of σ. We discuss this further in Section 3.3 when selecting our noise intensity values. We can however, qualitatively compare their relative impacts to develop some level of comparison. As shown in 4.2 we can observe approximately the same level of degradation for $\sigma = 0.1$ in AWGN as $\sigma = 10$ for Brownian noise, for instance we see VGG-16's top-1 FP32 and INT8 accuracy is 0.7777 and 0.7795 respectively in AWGN and 0.7795 and 0.7756 respectively under Brownian noise.

Lastly, we examine highly structured noise in the form of vertical and horizontal black out occlusions, or streaks, applied within a pre-defined bounding box of the classification target within each image, as shown in Figure 2.4. We generate these as having constant width such that increasing the number of streaks increases the amount of the image covered. The study of complete regions of occlusion is relevant in fields such as satellite imagery where it is not uncommon to experience sensor malfunctions or data corruption during transmission or due to solar radiation, resulting in rows or columns of dead pixels. We control the intensity or degree of perturbation induced by such occlusions by adjusting the number of streaks present in the class' bounding box region. It is expected that comparatively, such occlusions will significantly degrade performance. This is because CNNs rely on filters to extract features such as edges from images and we are injecting points of high contrast, which may cause irrelevant feature activation and false edge identification. Moreover, streaks may cover important or distinguishing features of a target class such as its engine or wing structure, again, confusing the network. The question here then becomes one of robustness, that is, when faced with a total loss of information how does the network rely on other features to identify a class.



(a) AWGN with $\sigma = 0.3$ (far left) and $\sigma = 0.6$ (left). Brownian noise with $\sigma = 40$ (right) and $\sigma = 70$ (far right).



(b) Vertical occlusions with 3 (far left) and 5 (left) streaks. Horizontal occlusions with 3 (right) and 5 (far right) streaks.

Figure 2.4: Demonstrations of various levels of studied noise.

2.3 Degradation Metrics

We consider several metrics to characterize the degradation of a given INT8 model with respect to its FP32 counterpart. Firstly, we consider accuracy and F1. Recall that generally, a given classifier can yield any of the following outputs:

- True positive (TP) - the classifier correctly identifies an input as belonging to its true class
- True negative (TN) - the classifier correctly identifies an input as not belonging to an incorrect class
- False positive (FP) - the classifier wrongly identifies an input as belonging to an incorrect class
- False negative (FN) - the classifier fails to identify an input as belonging to its true class

Accuracy is a measure of the proportion of correctly classified instances among the total instances. It provides a clear indication of the overall immediate performance of the model. In the context of model quantization, comparing the accuracy of the INT8 model to that of the FP32 model helps to quantify the drop in absolute classification performance and is calculated as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.16}$$

Specifically, we examine both top-1 accuracy and top-5 accuracy. Top-1 accuracy considers a prediction correct if the model output matches the true label, and top-5 accuracy considers a prediction correct if the true label is amongst the model's top-5 highest output probabilities. The utility in examining top-5 is that it reflects how often a model is essentially *close* to being correct, even if its not exact. The notion of top-5 accuracy originates from ImageNet [8], where within the broader dataset there existed several groupings of related sub-classes with subtle differences. Consequently, a more nuanced metric that captures if a model was close to being correct was desired. This is particularly relevant in our study of FGVC where the distinctions between classes are very subtle - it allows us to see if a model is *near correct* and thus permits some ambiguity. Functionally, this may indicate that a model is learning some important features for a given class but needs further training or optimization to satisfy the fine-grained subtleties between similar classes e.g. the wing structure between a Boeing 737 and a Boeing 777. However, accuracy does not give insight into the type of errors the model is making. To examine the specific types of errors, we can employ precision and recall and aggregate them with the F1 score. Precision is the ratio of true positive predictions to the total predicted positives, indicating how many of the predicted positives are actually positive, that is:

$$precision = \frac{TP}{TP + FP} \tag{2.17}$$

Recall is the ratio of true positive predictions to the total actual positives, indicating how many of the actual positive instances are captured by the model, or:

$$recall = \frac{TP}{TP + FN} \tag{2.18}$$

We then compute the F1 score to understand how balanced the model is in terms of its error modes by computing the harmonic average of precision (avoiding false positives) and recall (avoiding false negatives):

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{2.19}$$

Accuracy and F1 alone do not offer insight into the *sensitivity or confidence* of a model's predictions, meaning they fail to reflect how confident the model is about its decisions. This omission can have significant implications, as models with the same accuracy or F1 score might have vastly different levels of reliability and robustness, particularly against unexpected or random perturbations.

To analyze sensitivity and robustness we examine the class output probabilities using the Kullback-Leibler KL divergence. KL divergence, introduced in [1], quantifies how close an approximate distribution Q is to the true or baseline distribution P [38]. Mathematically it is defined as:

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{2.20}$$

Where i is the number of possible states. In our context, we wish to compare the divergence of an INT8 quantized model's output class probabilities to its full-precision counterpart, given K output classes. It is asymmetric, i.e. $(D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P))$, and always non-negative, reaching zero if and only if P and Q are identical. We denote the INT8 and FP32 models' class probability distributions as P_{int8} and P_{fp32}. Thus we get:

$$D_{KL}(P_{\text{fp32}} \parallel P_{\text{int8}}) = \sum_{k=1}^{K} P_{\text{fp32}}(k) \log \frac{P_{\text{fp32}}(k)}{P_{\text{int8}}(k)} \tag{2.21}$$

We demonstrate this by synthetically generating a standard normal distribution and computing its KL divergence with other distributions of varying standard deviations, as shown in Figure 2.5. We can readily observe that as we increase the standard deviation, the KL divergence increases. Intuitively, we are measuring the additional information required to encode data points from a true distribution using an approximate distribution, i.e., it is the cost of assuming Q , given that the true distribution is P [1].

In our study, this effectively allows us to measure the information loss introduced by quantization, as well as with respect to input perturbations. If we observe similar accuracy and/or F1 scores but high KL divergence, it may suggest that while the quantized model did not lose accuracy, it is less confident in its predictions. For example, consider a FP32 model that yields the following output probabilities: class

1: 0.7, class 2: 0.2, class 3: 0.1, and an INT8 model that yields: class 1: 0.5, class 2: 0.4, class 3: 0.1, where the true label is class 1. Both models correctly classified class 1 as the true label, however, they have markedly different confidences in their prediction. Moreover, examining KL divergence under different perturbation regimes can guide targeted optimization in quantization parameter selection, or even the applicability of quantization entirely. It is important to recognize, however, that this metric assumes the FP32 model is the *true* baseline distribution. We contend that this is a fair characterization as, in reality, it is the theoretical *best* we can do in terms of capacity for information embedding per parameter. It is true that the baseline in this case is then dynamic, in the sense that the distribution of the FP32 model will change with perturbation. Given that both the FP32 and INT8 models are tested on the exact same input with the exact same perturbation, we are then answering the question *under perturbation level X, how does our quantized model deviate in its probabilities from what would otherwise be outputted given no quantization?*.

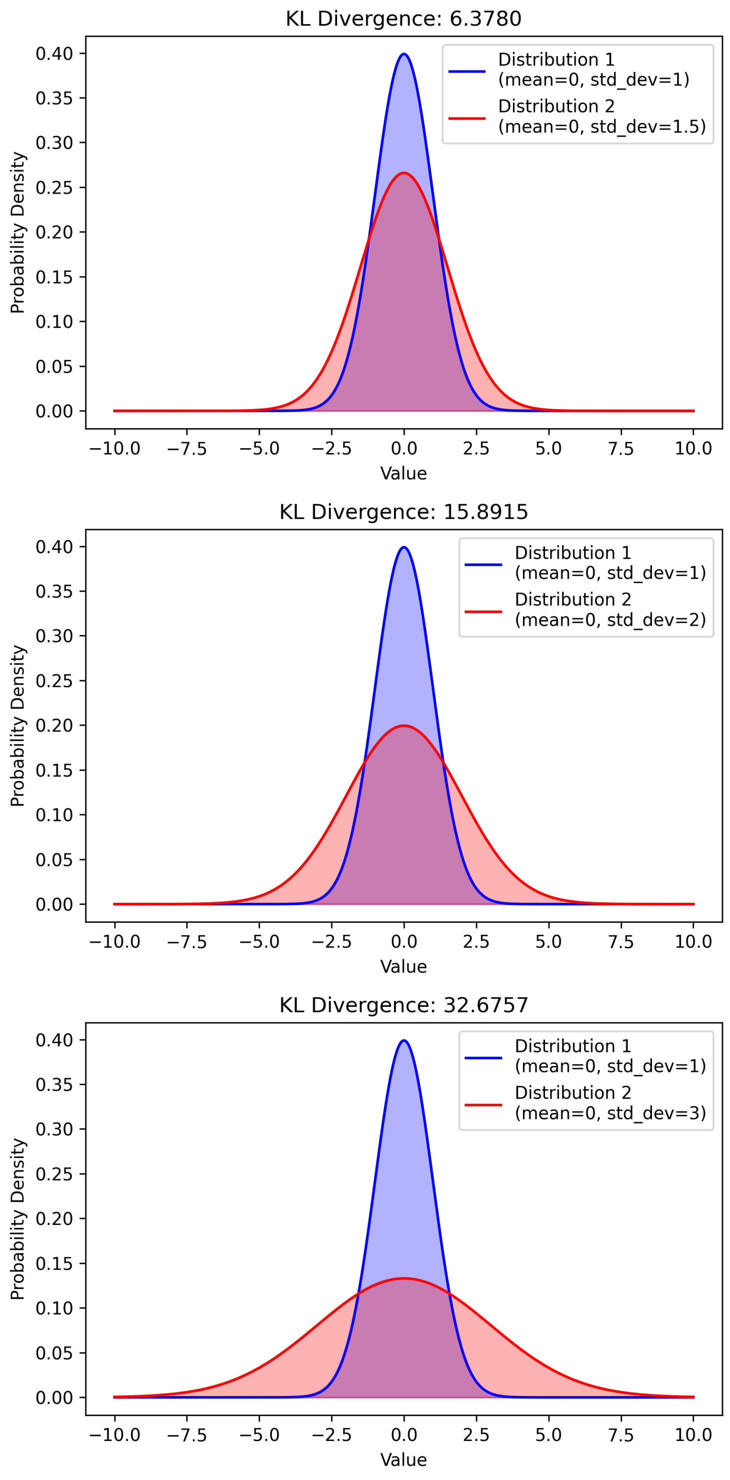


Figure 2.5: KL divergence for standard normal distribution with distributions of varying standard deviations.

Chapter 3

Methods

3.1 Model Selection and Dataset

To fully assess the impacts of quantization on a given model, we select three SOTA models, all of various sizes and architectural features. In order of largest to smallest, we study VGG-16 [39], ResNet-18 [15], and SqueezeNet1_1 [17]. These models are commonplace in the current CNN landscape and often used as SOTA benchmarks in conjunction with datasets like MNIST, CIFAR, and ImageNet. More importantly, however, is their differing size, with VGG-16 being the largest model in terms of parameters and memory footprint and SqueezeNet1_1 being the smallest, designed specifically for use on embedded platforms. Moreover, each model implements unique design philosophies and architectural features; VGG-16 is designed for depth, ResNet implements skip connections between layers, and SqueezeNet1_1 uses an expand/-compression scheme. What follows is a brief discussion of each of these models.

VGG-16

VGG, developed by researchers at the University of Oxford in 2015, is a CNN predicated on the notion of network depth paired with smaller filter sizes of 3×3, a stark departure from other SOTA models like AlexNet's 11×11 filter size [19]. VGG uses these 3×3 filters in each convolutional layer, paired with a ReLU activation function. In addition to these convolutional layers, five max-pooling layers are added, each with a 2×2 pixel window [39]. Following the convolutional layer and max-pooling stack, two 4096-channel fully-connected layers are implemented, followed by a final n-channel fully-connected layer and softmax activation function, where n is the number of output classes. There are several configurations of VGG, ranging from 11 convolutional layers to 19 convolutional layers. We study VGG-16 as it has less than a 1% top-1 error discrepancy with VGG-19 at a smaller memory footprint [39].

This VGG-16 configuration has 13 weight layers (convolutional layers), followed by the above fully-connected layers [39]. See Figure 3.1a for a diagram of VGG-16's architecture and Table 3.1 for a summary of its relative size.

ResNet-18

Developed at Mirocsoft in 2015, the ResNet family of CNNs seeks to address the problem of vanishing gradients[1]. ResNet begins with a 7×7 convolutional layer followed by a 3×3 max-pooling layer followed by a series of convolutional layers with ReLU activations [15]. There are several configurations of ResNet, ranging from 18-layer to 152-layer. We study the 18-layer configuration shown in Figure 3.1b, ResNet-18, to serve as the intermediary model between larger networks like VGG-16 and smaller networks like SqueezeNet1_1. ResNet-18 uses eight residual blocks, where each block is comprised of 2 convolutional layers with ReLU activations followed by an average-pooling layer and finally an n-channel output layer with the softmax activation function [15]. See Figure 3.1b for a block diagram of this architecture and Table 3.1 for a summary of its relative size and benchmark performance. ResNet differs from conventional networks in its introduction and use of shortcut connections. Consider the input to an above-mentioned block or grouping of convolutional layers as x. Shortcut connections skip a grouping of layers and add the previous block's x to the current grouping's output. That is, for a given residual block, its output is $F(x) + x$. These shortcut connections alleviate the vanishing gradient problem, thereby allowing for comparatively high layer counts - as many as 152.

SqueezeNet1_1

SqueezeNet1_1 is the smallest of the three chosen networks, specifically designed for applications in embedded systems and edge applications while still maintaining levels of accuracy comparable to its contemporaries [17]. SqueezeNet1_1's main building block is the fire module: a squeeze convolution layer with 1×1 filter size feeding into

[1]The vanishing gradients phenomenon occurs during training when the gradients of the loss function with respect to each weight diminish as they are propagated backward through the network's layers. This diminishment is primarily caused by the repeated multiplication of gradients that are less than 1 through the deep network structure, particularly when using activation functions like sigmoid or tanh. Since the updates to the weights in each layer depend on these gradients, the process can slow down or completely stagnate the network's learning ability.

an expand layer that is a mix of 1×1 and 3×3 filters [17]. The squeeze layer acts as a bottleneck to reduce the depth of the data passing through the network, thereby decreasing the number of parameters and subsequent resource overhead. The expand layer's 1×1 filters function to increase the dimensionality from the compressed output of the preceding squeeze layer, adding depth to the feature maps. Concurrently, the 3×3 filters in the expand layer work on the expanded feature maps to capture spatial patterns from the input, such as edges and their orientation. A diagram of SqueezeNet1_1's architecture is given in Figure 3.1c. SqueezeNet1_1's parameter count and benchmark performance are provided in Table 3.1.

An important quantity when considering a network, specifically its relative size, is the number of operations it must perform during a forward pass or inference, typically measured by the number of floating point operations (FLOPS). In the case of neural networks, a floating point operation occurs in the previously discussed MAC block. Higher FLOPS induces hardware overhead and thus serves as a bottleneck. Table 3.1 compares several quantities for the three considered networks alongside their respective benchmark top-1 accuracies on the ImageNet dataset. As expected, accuracy degrades with decreasing parameter counts and FLOPS.

Table 3.1: Details of studied models. All data retrieved from PyTorch documentation [32].

Model	Parameter Count (millions)	Size (MB)	GFLOPs	ImageNet Top-1 Acc.
VGG-16	138.4	527.8	15.47	71.592
ResNet-18	11.7	44.7	1.81	69.758
SqueezeNet1_1	1.2	4.7	0.35	58.178

To study fine-grained visual classification FGVC, we select the fine-grained visual classification aircraft (FGVC-A) dataset [26]. Many well-known FGVC datasets exist - for example, natural species [6], birds [16], or flowers [29]. However, as the dataset authors note, aircraft as a classification target offer several unique aspects. Firstly, aircraft vary significantly depending on their size (hobbyist project planes to large transport aircraft), purpose (commercial, pleasure, or military), and technology (turbine propulsion, propeller, glider, etc.), all of which yield different structural features such as the wing shape and size, fuselage style, landing gear/wheels, and engine mounting. Another interesting feature of FGVC-A is that planes like airliners and military aircraft are often reused by different organizations and have slight modifications such as branding and camouflage while still belonging to the same class. The dataset provides several classification tasks and labels, including aircraft model (most specific), variant, family, and manufacturer (least specific). This thesis studies the family classification task, in which 70 labels are used for different families. The families dataset groups together similar model variants and is considered intermediate difficulty [26]. Examples of families include Boeing-737, which includes variants like 737-200, 737-300, etc. [26]. While the variants dataset is balanced - there are 100 variants, with each variant class having 100 images -, the families dataset, a sub-set of variants, is not balanced, as shown below in Figure 3.3. We can see that the class "Boeing 737" dominates, which is expected, as there is a comparatively large amount of variants within the Boeing 737 family that are being grouped into this class[2]. See Figure 3.2 for example imagery from the dataset.

3.2 Quantization Scheme

We know describe our quantization methods and the specific quantization parameters used. In general, we follow the recommendations outlined in [28] and perform PTQ on each model's weights and activations according to the parameters given in Table 3.2. Python code for this quantization scheme's implementation is provided in Appendix A.

Table 3.2: Quantization parameters. Note that quantization range, scale, and zero point are learned parameters during calibration with observers.

Parameter	Weights	Activations
Observer	Min/max	Histogram
Bit width	8	8
Initial Min.	-127	0
Initial Max.	128	255
Initial Scale	0.1	0.1
Initial Zero Point	0.0	0.0
Symmetry	Symmetric	Affine
Resolution	Per-channel	Per-tensor

We can verify the validity of our implementation by printing out and comparing

[2]The Boeing 737 family class contains the variants 737-200, 737-300, ..., 737-900.

the FP32 and INT8 model architectures using PyTorch's torchsummary package [32]. For example we can see the changes to ResNet-18's first convolution/pooling/activation layer. Note the inclusion of the min/max observer for the weights and histogram observer for the activations. Moreover, we can see the inclusion of the fake quantize modules containing the quantized weights.

Table 3.3: Summary of FP32 ResNet-18's first convolutional layer architecture.

Layer	Output Shape
Conv2d-1	[-1, 64, 112, 112]
BatchNorm2d-2	[-1, 64, 112, 112]
ReLU-3	[-1, 64, 112, 112]
MaxPool2d-4	[-1, 64, 56, 56]

Table 3.4: Summary of INT8 ResNet-18's first convolutional layer architecture.

Layer	Output Shape
HistogramObserver-1	[-1, 3, 224, 224]
EQLearnableFakeQuantize-2	[-1, 3, 224, 224]
PerChannelMinMaxObserver-3	[-1, 3, 7, 7]
EQLearnableFakeQuantize-4	[-1, 3, 7, 7]
BatchNorm2d-5	[-1, 64, 112, 112]
HistogramObserver-6	[-1, 64, 112, 112]
EQLearnableFakeQuantize-7	[-1, 64, 112, 112]
MaxPool2d-8	[-1, 64, 56, 56]

3.3 Experimental Setup and Procedure

We now outline the procedure for quantifying the performance of the models introduced in Section 3.1, according to the metrics discussed in Section 2.2.

Model Training

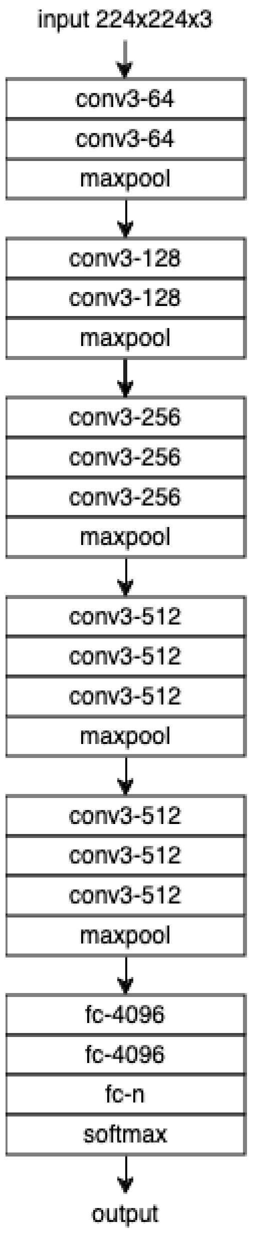
Models were downloaded from PyTorch's model zoo [32], with pre-trained weights for ImageNet [8]. Each model was adjusted to have 70 output neurons, in accordance with the FGVC-aircraft families classification task. Models were trained for 250 epochs on a training set of shuffled 3333 images, using the cross-entropy loss function to measure the prediction error. The stochastic gradient descent optimizer was employed with a learning rate of 0.001 and a momentum of 0.9 to update the model parameters during training. Additionally, a learning rate scheduler was applied to decrease the learning rate by a factor of 0.1 every 50 epochs.

Perturbation Intensities

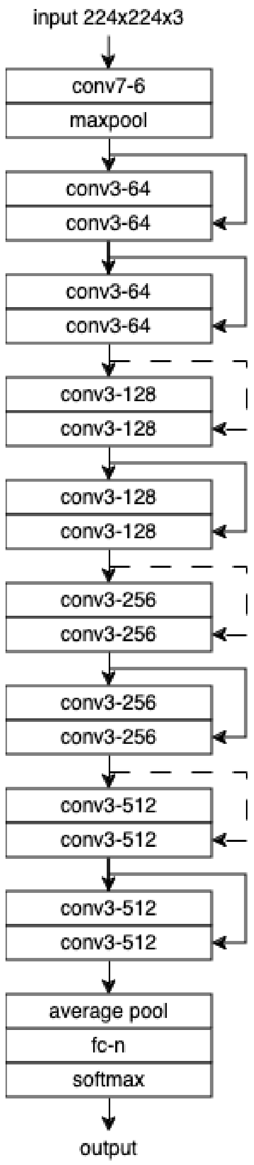
We experiment the performance of each model under each of the perturbations discussed in Section 2.2. As noted, each of these perturbations has its own respective intensity parameter. For AWGN and Brownian noise it is standard deviation, and the number of streaks for vertical and horizontal occlusions. In each case, the intensity range was qualitatively selected to best capture the full spectrum of degradation, as is shown in the accuracy/F1 plots in Section 4.2. For AWGN we study standard deviations ranging from 0 to 1 with a step size of 0.1 from 0.1 to 0.6, and then 0.2 from 0.6 to 1.0 as beyond 0.6 model performance plateaus. For Brownian noise we study standard deviations ranging from 10 to 80 with a step size of 10. For vertical and horizontal occlusions we select streaks ranging from 2 to 6 with step sizes of 1, where each streak is 2% of the size of a given target's bounding box. Note that the inclusion of even 1 streak yielded significant degradation across all metrics, and as more streaks were added, performance quickly degraded to near 0 across all metrics. Also, as discussed in Section 2.2 the Brownian perturbation regime required comparatively higher standard deviations than AWGN to yield any noticeable degradation. This is expected, as Brownian noise is smoother and more structured than AWGN, owing to its $1/f^2$ spectrum, making it less disruptive to the image's overall structure and features when compared to AWGN's direct pixel-by-pixel variations. For each perturbation scheme each model's FP32/INT8 pair were tested on the full test set of 3333 images. We demonstrate the impacts sample size has on performance in Section 4.1.

Degradation Metrics Computation

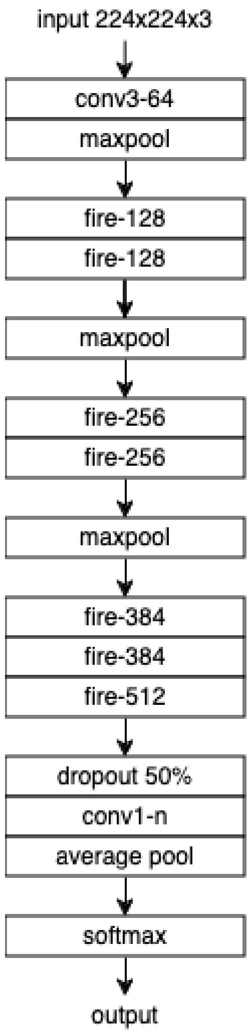
Recall from Section 2.3, we are interested in computing 4 primary metrics, top-1 accuracy, top-5 accuracy, F1-score, and KL divergence. Top 1 accuracy, top 5 accuracy, and F1 score are easily computed using Scikit-learn [33]. To compute KL divergence we store the 70 class output probabilities for each model pair and compute the KL divergence between them, on a per image basis, for each perturbation level. We then compute the average KL divergence across the entire test set of 3333 images, for each perturbation level, resulting a in a single scalar averaged measure for each model pair. Code for this process is presented in Appendix B.



(a) VGG-16



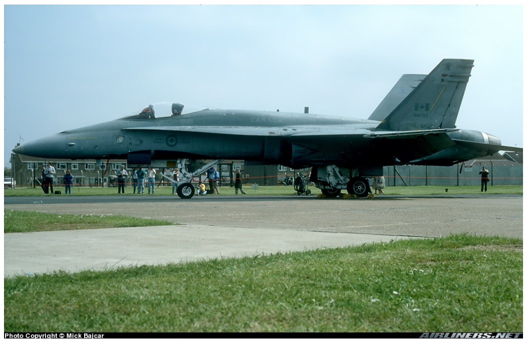
(b) ResNet-18



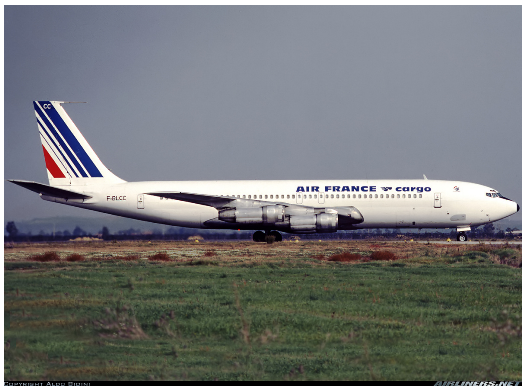
(c) SqueezeNet1_1

Figure 3.1: Architecture diagrams of the studied models: (a) VGG-16, (b) ResNet-18, and (c) SqueezeNet1_1. conv3-64 denotes a convolutional layer with kernel size 3×3 and an output of 64 features; fc-n denotes an n-channel fully-connected layer; conv1-n denotes a 1×1 kernel with n output features.

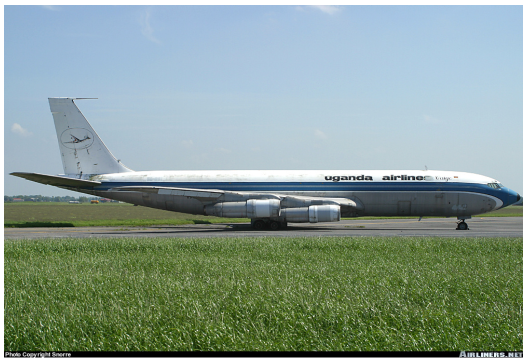
(a) Cessna-172



(b) F/A-18



(c) Boeing-707



(d) Boeing-707

Figure 3.2: Sample imagery from the FGVC-A dataset. Note that (c) and (b) belong to the same class but have different branding.

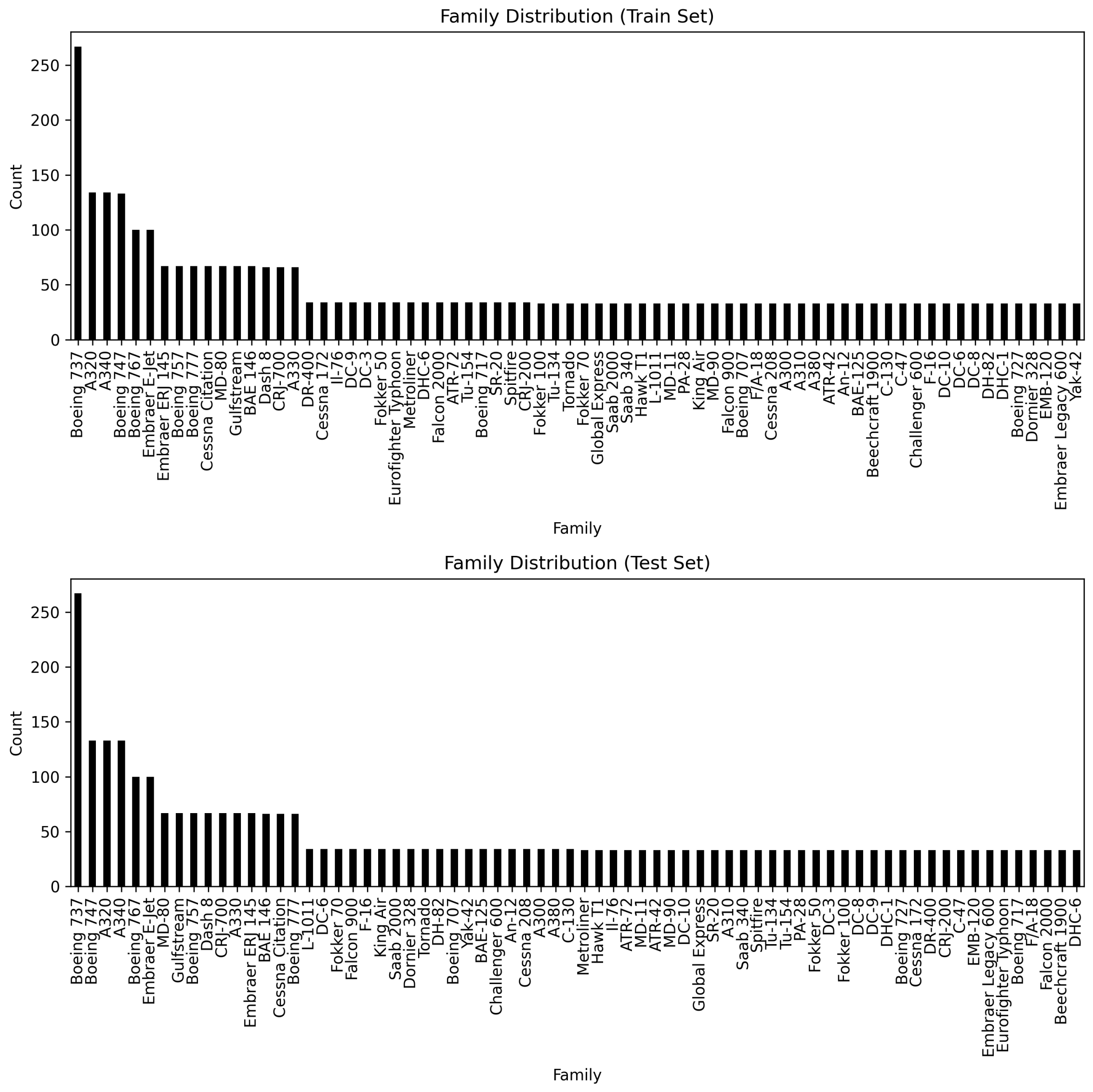


Figure 3.3: Class counts for FGVC-Aircraft family dataset.

Chapter 4

Results

4.1 Baseline Results

We now present the results of our experiments. We begin by examining the unperturbed results to establish a baseline. Table 4.1 presents accuracy and F1 scores for each model FP32 and INT8 pair. Immediately, we can see that VGG-16 performs the best in all categories and SqueezeNet1_1 the worst. This expected result matches the general trend presented in Table 3.1 and supports the notion that sheer parameter count manifested as layer depth yields higher accuracy. Moreover, the relative error rates between each model pair match the results presented in [28]. As expected, we can see that top-1 accuracy is comparatively more degenerate than top-5 accuracy across all models - SqueezeNet1_1 exhibits a 14% top-1 accuracy drop compared to VGG-16 but only a 6% drop in top-5 accuracy. This is indicative of the nature of FGVC where it is likely that smaller models like SqueezeNet1_1 lack the expressive power to capture the subtleties between similar classes but can make a *near* correct prediction. We also see that F1 - the average of a model's ability to avoid false positives (precision) and false negatives (recall) - follows the same trend as top-1 accuracy. Most notable, however, is that we observe very little degeneration in model performance across all three metrics post-quantization. This result is congruent with [28] [40] [41] and confirms that our quantization scheme is implemented and performing as expected. Table 4.2 presents KL divergences for each model pair. Interestingly, ResNet-18 exhibits the *lowest* KL divergence by an order of magnitude, and VGG-16 the highest. While all relatively low, these results may reflect each model's architecture. For instance, ResNet-18's skip connections, as discussed in Section 3.1, may aid in mitigating quantization error propagation where VGG-16's depth without such connections may exacerbate quantization error propagation.

Table 4.1: Baseline unperturbed results.

	SqueezeNet1_1		ResNet-18		VGG-16	
Metric	FP32	INT8	FP32	INT8	FP32	INT8
Top-1 Accuracy	0.6268	0.6250	0.7204	0.7192	0.7843	0.7828
Top-5 Accuracy	0.8866	0.8866	0.9241	0.9256	0.9487	0.9475
F1 Score	0.6287	0.6268	0.7153	0.7145	0.7835	0.7828

Table 4.2: Baseline unperturbed KL divergences for each model pair.

Model Pair	KL Divergence
SqueezeNet1_1	0.0138
ResNet-18	0.0082
VGG-16	0.0182

4.2 Perturbed Results

4.2.1 AWGN

We begin by looking at the impacts of AWGN, presented in Figure 4.1, and Tables 4.3, 4.4 and 4.5. Across all models and metrics, we see steep degeneration beginning at $\sigma = 0.2$ and plateauing at $\sigma = 0.8$, indicating a point of potential ill-conditioning in this particular noise regime. As expected, top-5 accuracy is consistently higher than top-1 by a 20% - 30% margin, with the gap between them widening as noise levels increase. This result suggests that while the exact classification becomes more challenging, the correct class will likely remain among the top 5 predictions. Moreover, in line with our baseline observations, VGG-16 generally outperforms ResNet-18 and SqueezeNet1_1, and ResNet-18 typically outperforms SqueezeNet1_1, with the gap narrowing at higher noise levels. F1 scores follow similar trends to accuracy metrics but show a steeper decline with respect to noise, suggesting that both precision and recall are possibly more severely impacted by noise than accuracy metrics. Most important to this study, however, is that in all cases, we see no significant divergence in performance curves between each FP32/ INT8 model pair, directly supporting the notion that quantized models, under AWGN are *just as* robust, according to accuracy and F1 metrics, irrespective of *overall* losses. That is, while a model may lose accuracy with increasing noise, the quantized model does not exhibit ill-conditioning *relative* to its FP32 counterpart. Figure 4.3 and Table 4.6 presents the KL divergences under AWGN. Here, we witness interesting non-linear behavior - under AWGN, each model's INT8 representation exhibits peak divergence in its class output probabilities in the $\sigma = 0.4$ - 0.6 range. Also, in line with the trend in baseline KL divergences, VGG-16 demonstrates the highest divergence despite being less sensitive in its accuracy and F1 response curves. This suggests that while VGG-16 in its INT8 form is demonstrably robust according to accuracy and f1, its output distribution shifts from its FP32 form, potentially indicating decreased *overall* confidence or *misplaced* confidence. This result is particularly relevant in threshold-based systems where decisions are made based on some output probability threshold; this may induce increased false positives or false negatives. The drop in KL divergence after $\sigma = 0.6$ is most likely due to the model pairs reaching a saturation point in error, where, in effect, each model pair is equally weak and unable to make any meaningful predictions, which is evidenced in the accuracy plots where we can see sub-20% top-1 accuracy and sub-40% top-5 accuracy for both each quantized and full-precision model.

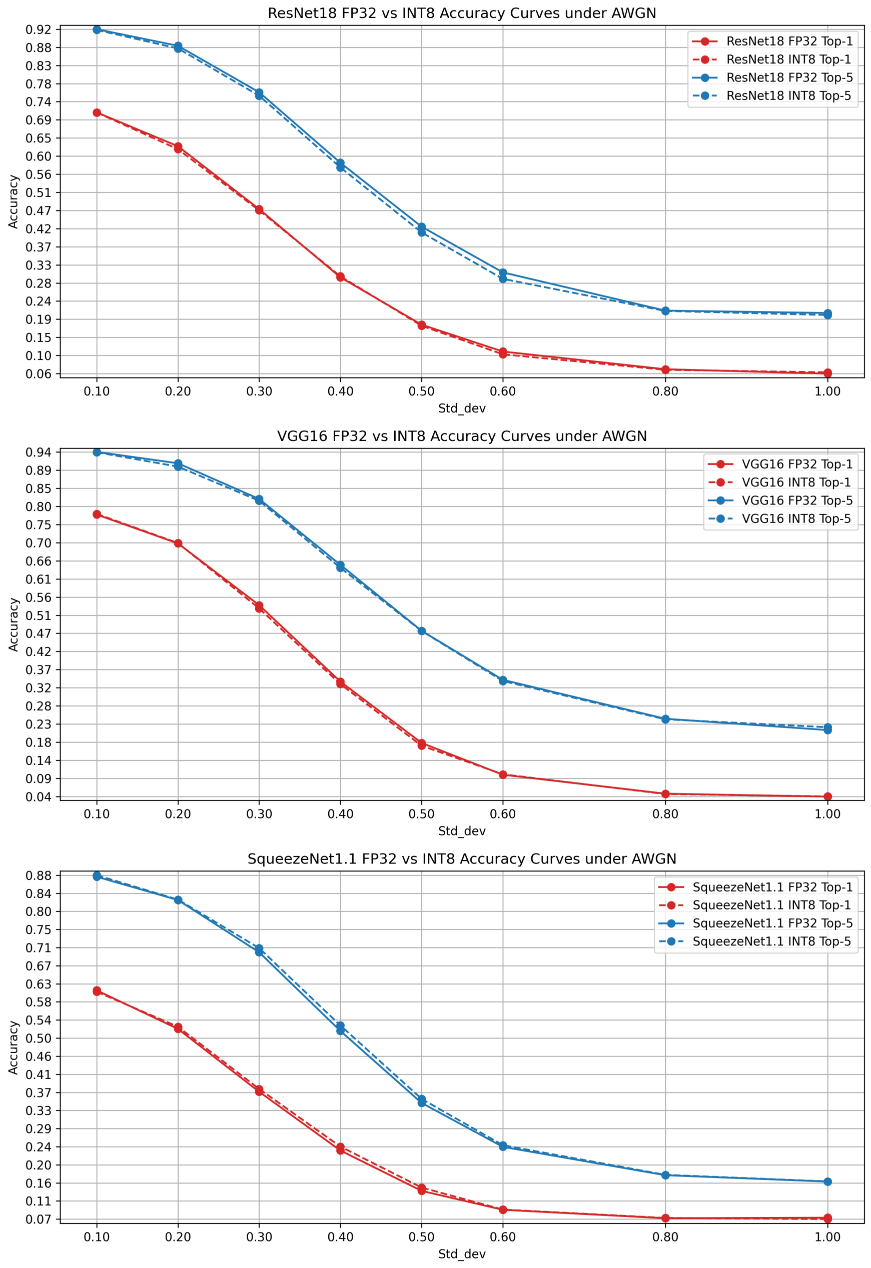


Figure 4.1: Top-1 and top-5 accuracies for each model's FP32/INT8 pair for varying levels of AWGN.

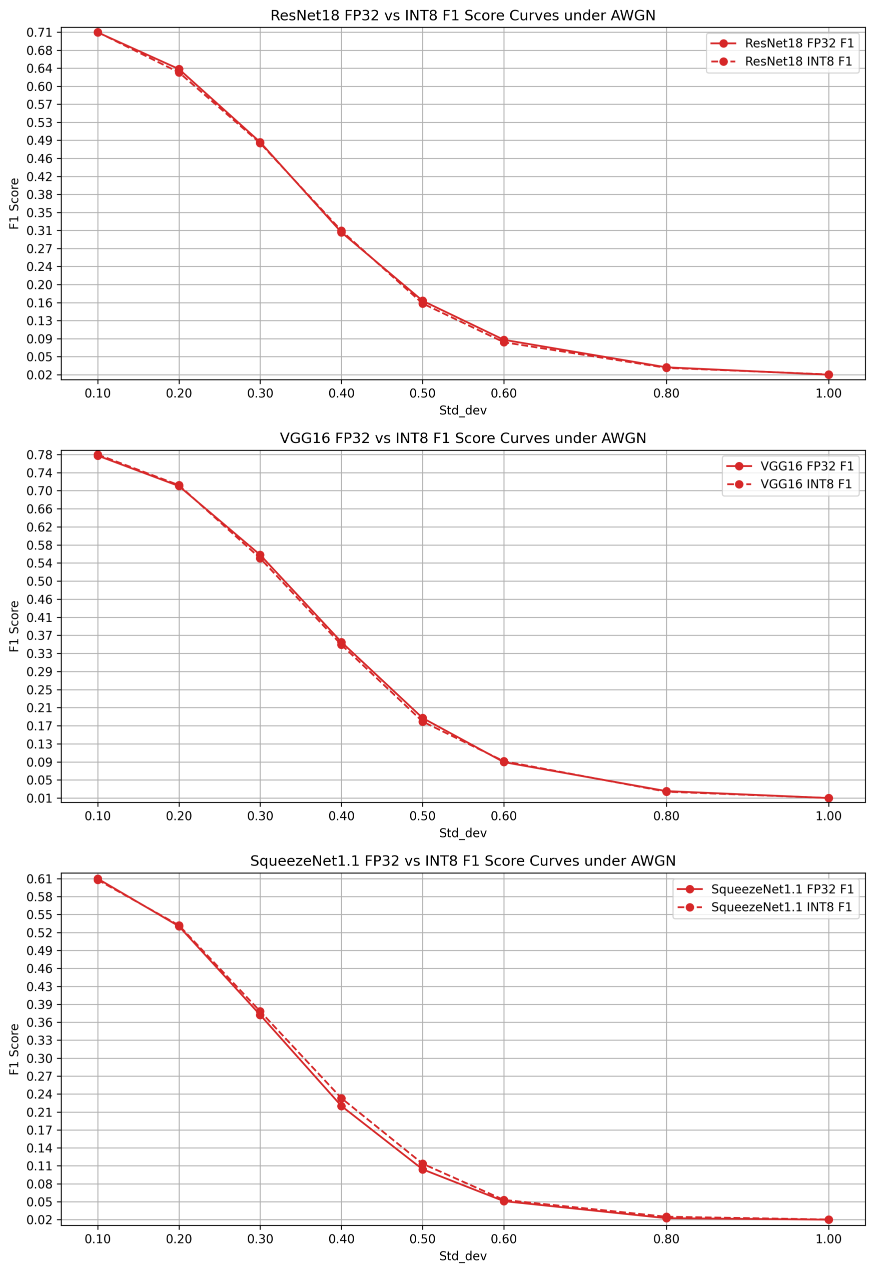


Figure 4.2: F1 scores for each model's FP32/INT8 pair for varying levels of AWGN.

Table 4.3: Tabulated top-1 accuracies for various levels of AWGN.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
0.1	0.6115	0.6082	0.7117	0.7117	0.7777	0.7795
0.2	0.5206	0.5257	0.6268	0.6193	0.7024	0.7033
0.3	0.3723	0.3789	0.4686	0.4656	0.5407	0.5323
0.4	0.2337	0.2430	0.2973	0.2997	0.3408	0.3345
0.5	0.1386	0.1464	0.1782	0.1755	0.1806	0.1734
0.6	0.0939	0.0945	0.1107	0.1038	0.0975	0.0987
0.8	0.0741	0.0747	0.0663	0.0648	0.0480	0.0471
1.0	0.0753	0.0720	0.0552	0.0585	0.0402	0.0408

Table 4.4: Tabulated top-5 accuracies for various levels of AWGN.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
0.1	0.8794	0.8830	0.9208	0.9190	0.9415	0.9406
0.2	0.8245	0.8254	0.8791	0.8725	0.9118	0.9031
0.3	0.7012	0.7111	0.7624	0.7537	0.8185	0.8137
0.4	0.5158	0.5287	0.5851	0.5734	0.6463	0.6385
0.5	0.3459	0.3555	0.4245	0.4101	0.4737	0.4734
0.6	0.2424	0.2463	0.3096	0.2934	0.3456	0.3426
0.8	0.1755	0.1764	0.2136	0.2127	0.2439	0.2424
1.0	0.1605	0.1608	0.2076	0.2025	0.2142	0.2217

Table 4.5: Tabulated F1 scores for various levels of AWGN.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
0.1	0.6149	0.6128	0.7126	0.7131	0.7792	0.7817
0.2	0.5314	0.5330	0.6380	0.6312	0.7106	0.7126
0.3	0.3758	0.3821	0.4893	0.4874	0.5554	0.5478
0.4	0.2159	0.2297	0.3061	0.3093	0.3589	0.3527
0.5	0.1047	0.1146	0.1667	0.1613	0.1876	0.1796
0.6	0.0489	0.0511	0.0875	0.0825	0.0882	0.0902
0.8	0.0190	0.0216	0.0315	0.0304	0.0229	0.0215
1.0	0.0164	0.0167	0.0165	0.0170	0.0072	0.0074

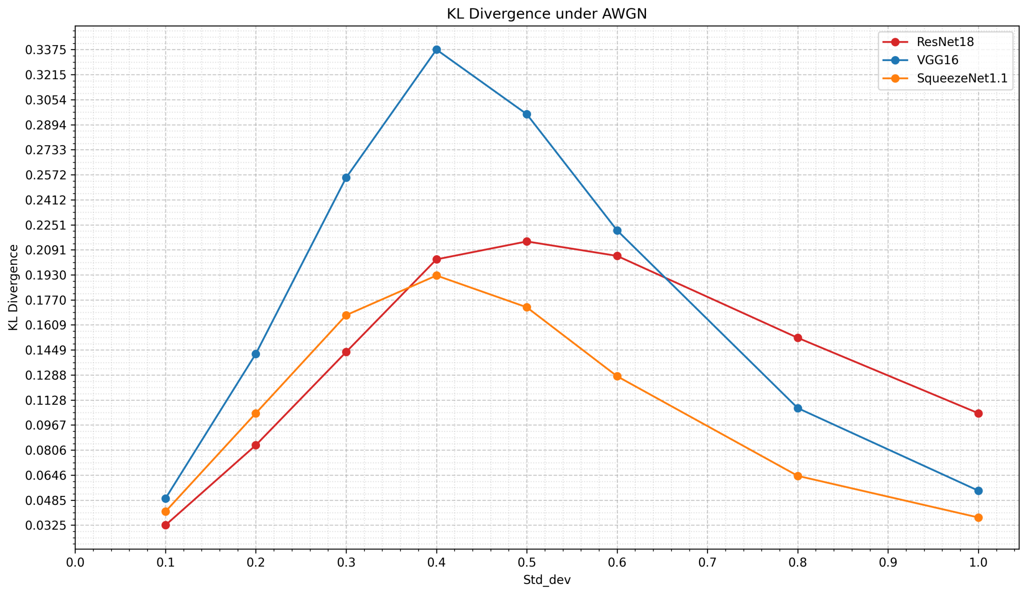


Figure 4.3: KL divergences for each model pair under various levels of AWGN.

Table 4.6: KL divergences for each model pair under various levels of AWGN.

Std_Dev	SqueezeNet1_1	ResNet-18	VGG-16
0.1	0.0412	0.0325	0.0496
0.2	0.1043	0.0838	0.1424
0.3	0.1672	0.1435	0.2554
0.4	0.1926	0.2030	0.3375
0.5	0.1722	0.2145	0.2961
0.6	0.1280	0.2052	0.2216
0.8	0.0641	0.1526	0.1076
1.0	0.0374	0.1043	0.0546

4.2.2 Brownian Noise

Next, we examine the impacts of varying levels of Brownian noise. Figure 4.4 and Tables 4.7 and 4.8 presents the top-1 and top-5 accuracies where we can see very different response curve compared to the AWGN results. It is clear that, as expected, the effects of Brownian noise are much more gradual compared to AWGN, as the spatial correlation of Brownian noise yields a much smoother noise pattern. Like AWGN, we see VGG-16 outperform all models as it maintains top-1 accuracy above 50% top-5 accuracy above 70% until $\sigma = 50$. Interestingly, we can see that Brownian noise seems to induce a relatively large error in the quantized model from the

FP32 model compared to AWGN, especially in VGG-16 in the σ = 10-50 range. In terms of F1, as shown in Figure 4.5 and Table 4.9, we also observe more significant discrepancies between the quantized and full-precision models, for instance, VGG-16 around 2% error in F1 at $\sigma = 20$. Also, as expected, the top-5 accuracy is consistently higher than the top-1 accuracy in all three models. Further, KL divergence, as shown in Figure 4.6 and Table 4.10 is much higher than AWGN and does not have the same peaking behavior shown in Figure 4.3 - instead we see a steady increase with noise across all three models. Like AWGN, VGG-16 exhibits the highest divergence, again suggesting a comparatively higher discrepancy in the model's confidence amongst output classes when making predictions. ResNet-18, in contrast, yields the lowest divergence. Overall, we can see that the impacts of Brownian are much more gradual; however, they induce significantly higher KL divergences when compared to AWGN.

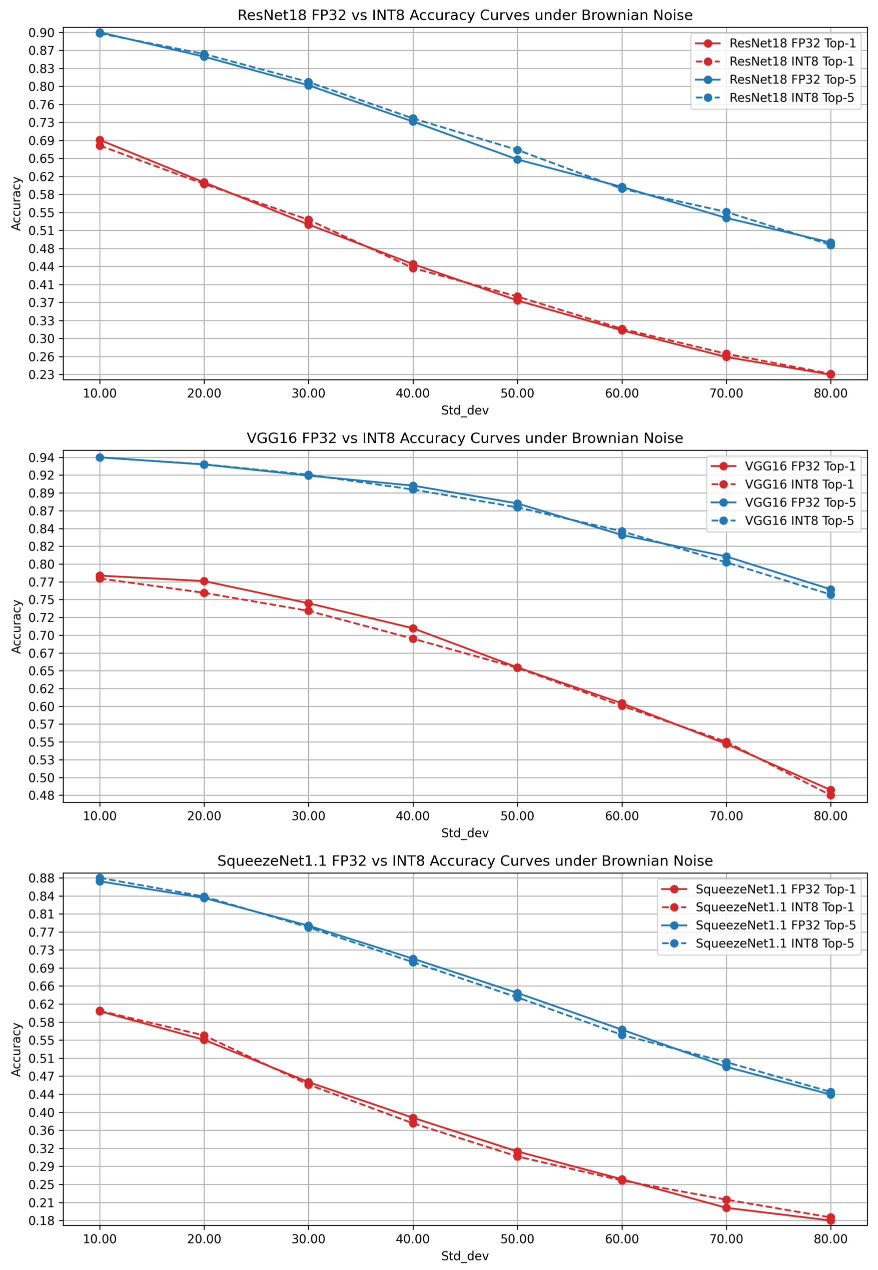


Figure 4.4: Top-1 and top-5 accuracies for each model's FP32/INT8 pair for varying levels of Brownian noise.

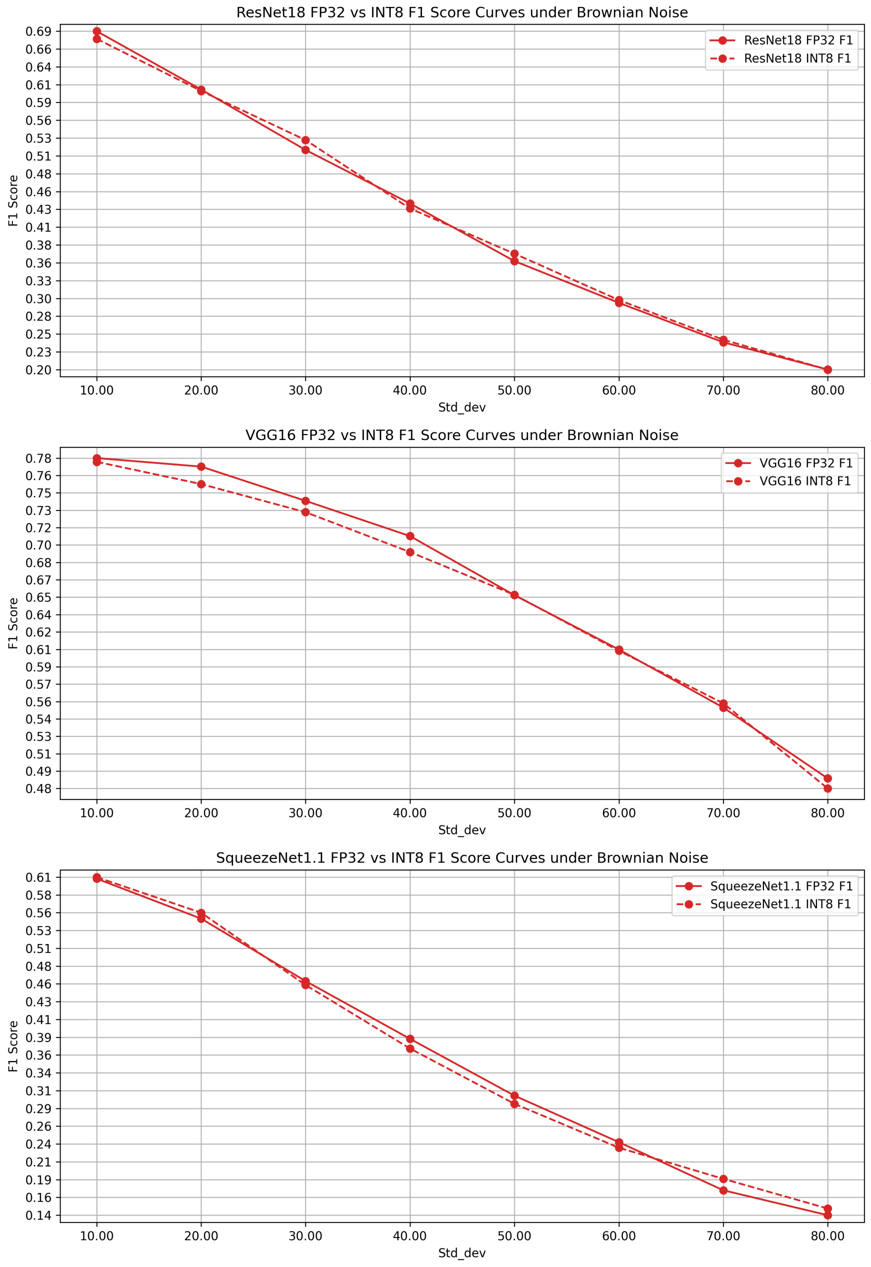


Figure 4.5: F1 scores for each model's FP32/INT8 pair for varying levels of Brownian noise.

Table 4.7: Tabulated top-1 accuracies for various levels of Brownian noise.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
10	0.6058	0.6067	0.6916	0.6805	0.7795	0.7756
20	0.5470	0.5560	0.6076	0.6043	0.7720	0.7558
30	0.4599	0.4548	0.5242	0.5335	0.7414	0.7309
40	0.3867	0.3753	0.4458	0.4380	0.7069	0.6925
50	0.3174	0.3072	0.3738	0.3816	0.6529	0.6523
60	0.2604	0.2577	0.3147	0.3177	0.6034	0.5998
70	0.2016	0.2184	0.2619	0.2682	0.5473	0.5503
80	0.1755	0.1818	0.2271	0.2286	0.4836	0.4764

Table 4.8: Tabulated top-5 accuracies for various levels of Brownian noise.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
10	0.8728	0.8800	0.9046	0.9025	0.9427	0.9427
20	0.8386	0.8416	0.8563	0.8617	0.9328	0.9331
30	0.7816	0.7783	0.7996	0.8062	0.9175	0.9187
40	0.7138	0.7066	0.7279	0.7342	0.9037	0.8983
50	0.6433	0.6343	0.6529	0.6718	0.8791	0.8737
60	0.5680	0.5572	0.5986	0.5950	0.8356	0.8407
70	0.4914	0.5011	0.5371	0.5491	0.8059	0.7978
80	0.4344	0.4398	0.4884	0.4836	0.7606	0.7534

Table 4.9: Tabulated F1 scores for various levels of Brownian noise.

	SqueezeNet1_1		ResNet-18		VGG-16	
Std_Dev	FP32	INT8	FP32	INT8	FP32	INT8
10	0.6062	0.6084	0.6883	0.6773	0.7793	0.7759
20	0.5505	0.5587	0.6044	0.6024	0.7716	0.7557
30	0.4634	0.4579	0.5176	0.5318	0.7404	0.7301
40	0.3828	0.3693	0.4407	0.4335	0.7083	0.6938
50	0.3036	0.2923	0.3578	0.3685	0.6546	0.6549
60	0.2387	0.2310	0.2979	0.3019	0.6051	0.6040
70	0.1716	0.1876	0.2411	0.2448	0.5523	0.5562
80	0.1369	0.1462	0.2020	0.2018	0.4878	0.4786

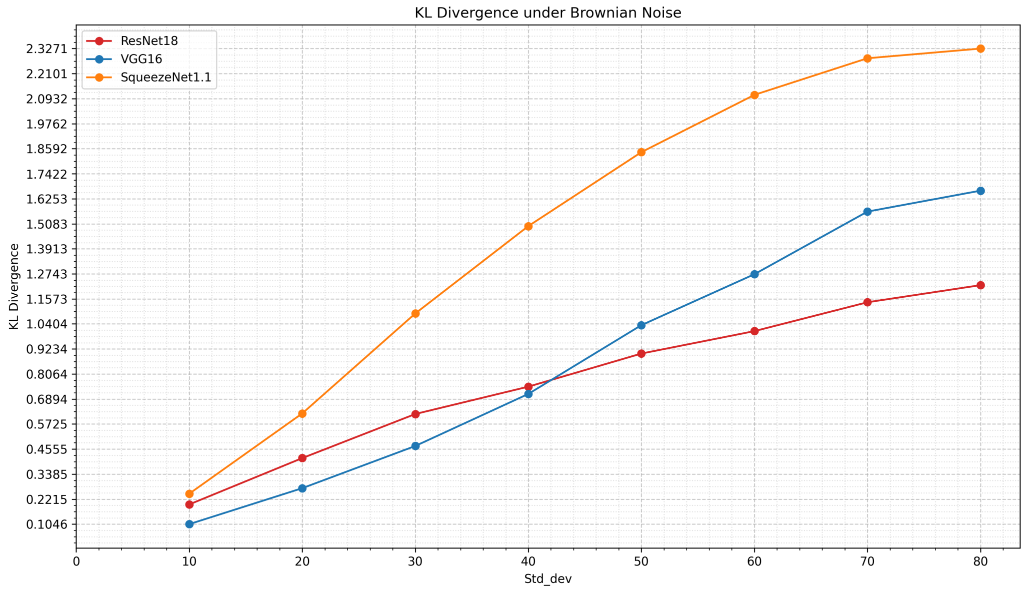


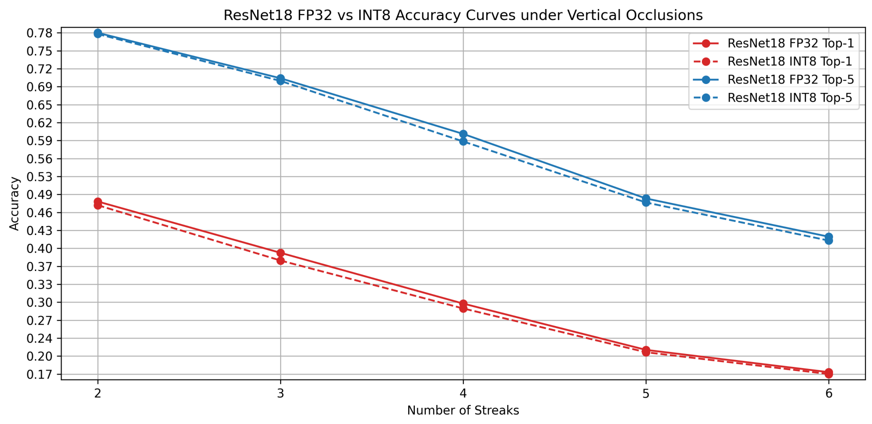
Figure 4.6: KL divergences for each model pair under various levels of Brownian noise.

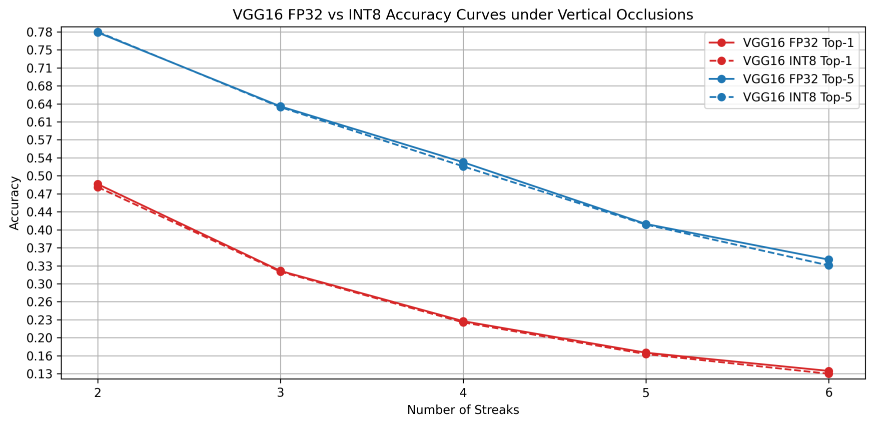
Table 4.10: KL divergences for each model pair under various levels of Brownian noise.

Std_Dev	SqueezeNet1_1	ResNet-18	VGG-16
10	0.2467	0.1967	0.1046
20	0.6221	0.4133	0.2727
30	1.0892	0.6193	0.4699
40	1.4975	0.7471	0.7132
50	1.8430	0.9018	1.0345
60	2.1109	1.0067	1.2726
70	2.2820	1.1416	1.5653
80	2.3271	1.2215	1.6631

4.2.3 Vertical Occlusions

We now look at the results for varying amounts of vertical occlusion, given in Figures 4.7, 4.8 and Tables 4.11, 4.12 and 4.13. In general, we see performance degradation in accuracy and F1 gradually dropping off, similar to Brownian noise, which is an expected feature as occlusions are another form of highly structured noise. However, unlike Brownian noise, the initial drop in accuracy from just two streaks is significant, with all models yielding less than 50% top-1 accuracy. Top-5 accuracy, however, is not as impacted until five streaks for ResNet-18 and VGG-16. On the other hand, SqueezeNet1_1 demonstrates considerably worse performance across all metrics. This is likely due to its comparatively small parameter space and lack of redundancy. Overall, we can see that in terms of accuracy and F1, ResNet-18 performs the best and exhibits the most minor degradation with increasing perturbation, contrary to previous results in which VGG-16 performed the best. Another interesting result is that top-1 and top-5 accuracies do not share the same general trend as F1. Instead, we see significant degradation in F1 immediately with two streaks and comparatively rapid decay, suggesting a higher rate of false positives/false negatives compared to true positives and true negatives. This result may suggest that, as we have an imbalanced dataset, models struggle more with minority classes as occlusion increases. With respect to quantization, we see the same general trend as other perturbation regimes, in that there is minimal divergence between each FP32/INT8 model pair, again, suggesting that performance degradation is a *macroscopic* phenomenon - a function of model architecture and overall size, not parameter-wise information capacity. Indeed, each model exhibits relatively low KL divergence with the max being 0.0636 for VGG-16 at two streaks, as shown in Figure 4.9 and Table 4.14. Moreover, the distribution divergence is relatively constant, with the most significant delta being just 0.0223 for VGG-16 between 2 and 6 streaks. However, it is notable that ResNet-18 exhibits the lowest KL divergence, which, paired with its performance in top-1, top-5, and F1, indicates it is comparatively robust against highly structured perturbation.





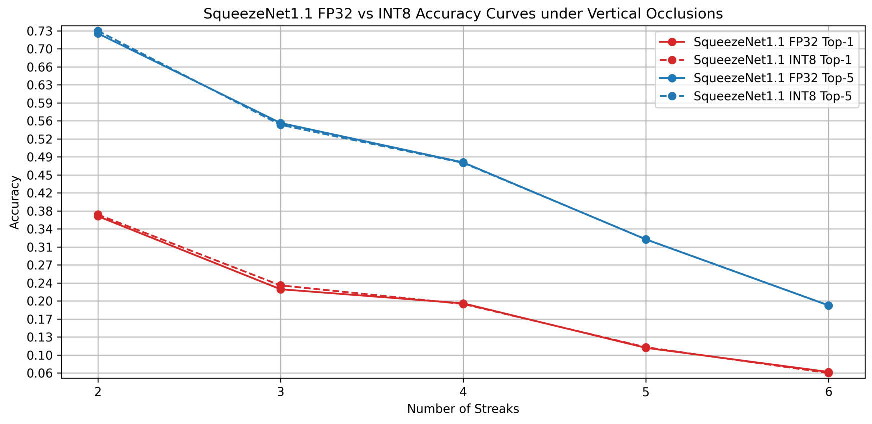


Figure 4.7: Top-1 and top-5 accuracies for each model's FP32/INT8 pair under various amounts of vertical occlusion.

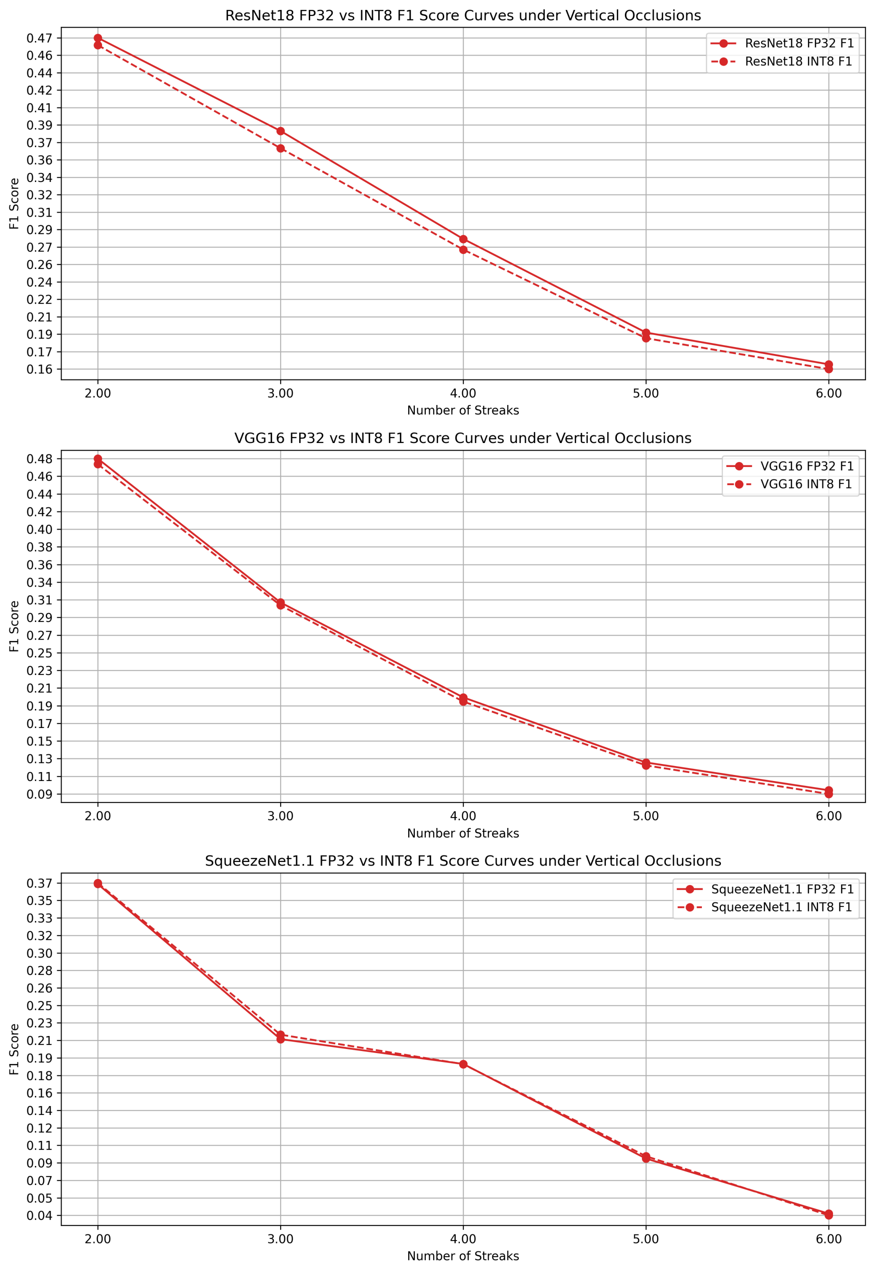


Figure 4.8: F1 scores for each model's FP32/INT8 pair under various amounts of vertical occlusion.

Table 4.11: Tabulated top-1 accuracies under various amounts of vertical occlusion.

	SqueezeNet1_1		ResNet-18		VGG-16	
Num. Streaks	FP32	INT8	FP32	INT8	FP32	INT8
2	0.3696	0.3729	0.4812	0.4746	0.4887	0.4830
3	0.2268	0.2343	0.3894	0.3756	0.3228	0.3213
4	0.1992	0.1980	0.2985	0.2898	0.2268	0.2244
5	0.1122	0.1134	0.2157	0.2115	0.1665	0.1638
6	0.0651	0.0630	0.1758	0.1725	0.1317	0.1263

Table 4.12: Tabulated top-5 accuracies under various amounts of vertical occlusion.

	SqueezeNet1_1		ResNet-18		VGG-16	
Num. Streaks	FP32	INT8	FP32	INT8	FP32	INT8
2	0.7273	0.7324	0.7831	0.7810	0.7786	0.7798
3	0.5521	0.5485	0.7018	0.6970	0.6373	0.6358
4	0.4749	0.4740	0.6022	0.5887	0.5305	0.5227
5	0.3243	0.3246	0.4866	0.4794	0.4125	0.4113
6	0.1953	0.1953	0.4182	0.4113	0.3444	0.3333

Table 4.13: Tabulated F1 scores under various amounts of vertical occlusion.

	SqueezeNet1_1		ResNet-18		VGG-16	
Num. Streaks	FP32	INT8	FP32	INT8	FP32	INT8
2	0.3689	0.3700	0.4727	0.4657	0.4807	0.4742
3	0.2134	0.2178	0.3841	0.3677	0.3117	0.3083
4	0.1885	0.1883	0.2814	0.2713	0.2002	0.1953
5	0.0937	0.0960	0.1924	0.1871	0.1237	0.1203
6	0.0384	0.0367	0.1622	0.1577	0.0911	0.0868

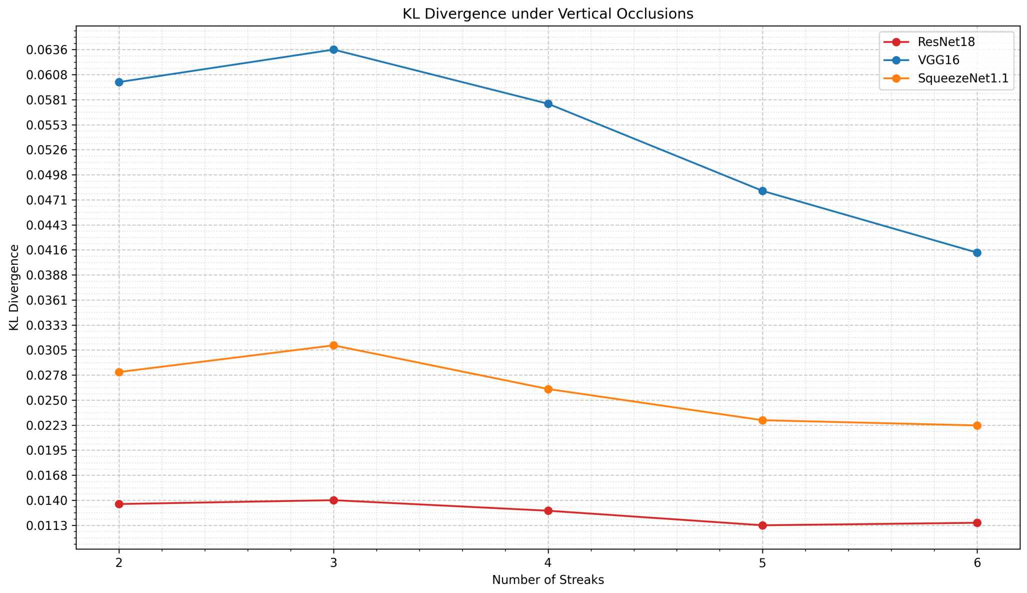


Figure 4.9: KL divergences for each model pair under various amounts of vertical occlusion.

Table 4.14: KL divergences for each model pair under various amounts of vertical occlusion.

Num. Streeaks	SqueezeNet1_1	ResNet-18	VGG-16
2	0.0281	0.0136	0.0600
3	0.0311	0.0140	0.0636
4	0.0263	0.0129	0.0576
5	0.0228	0.0113	0.0480
6	0.0223	0.0116	0.0413

4.2.4 Horizontal Occlusions

Lastly, we look at horizontal occlusions. Figure 4.10 and Tables 4.15 and 4.16 display the top-1 and top-5 accuracies, where we can see a similar result to the vertical occlusions and Brownian noise in that the degradation is gradual. Also similar is that we see an immediate drop off at just two streaks, again showing that generally, models are comparatively much more sensitive to highly structured occlusions compared to AWGN and Brownian noise. Also of note is that VGG-16 performs the best out of all three models across accuracy and F1, unlike the vertical occlusions. F1, as shown in Figure 4.11 and Table 4.17, is also similar to the vertical occlusion results, as we can see a steep decay congruent with the top-1 accuracy curve. Another interesting note is that we generally observe less degradation for the same number of streaks than vertical occlusions. This may be because the distinguishing features learned during training are dominated by horizontal edges such as the plane's fuselage, and introducing points of high contrast in the form of structured noise is much more disruptive, as they are guaranteed to intersect the plane's structure. Further, we see a widened discrepancy between top-1 and top-5 accuracies with horizontal occlusions compared to vertical, suggesting it is likely that generally, models are more likely to have the correct class in their top 5 predictions but struggle with exact classification. This result is likely exacerbated by the dataset's nature, in that such occlusion severely impairs the model's ability to distinguish between subtleties in the different classes. We can also see the same behavior as all other studied perturbations with respect to quantization in that there is minimal divergence between the INT8 and FP32 models across all metrics, with the max delta being 0.0069 for SqueezeNet1_1 at four streaks. Most unique to horizontal occlusions is that the KL divergence for all model pairs is near 0 and shows no clear trend. This indicates that the output distributions are similar under such perturbation and lack any notable discrepancy.

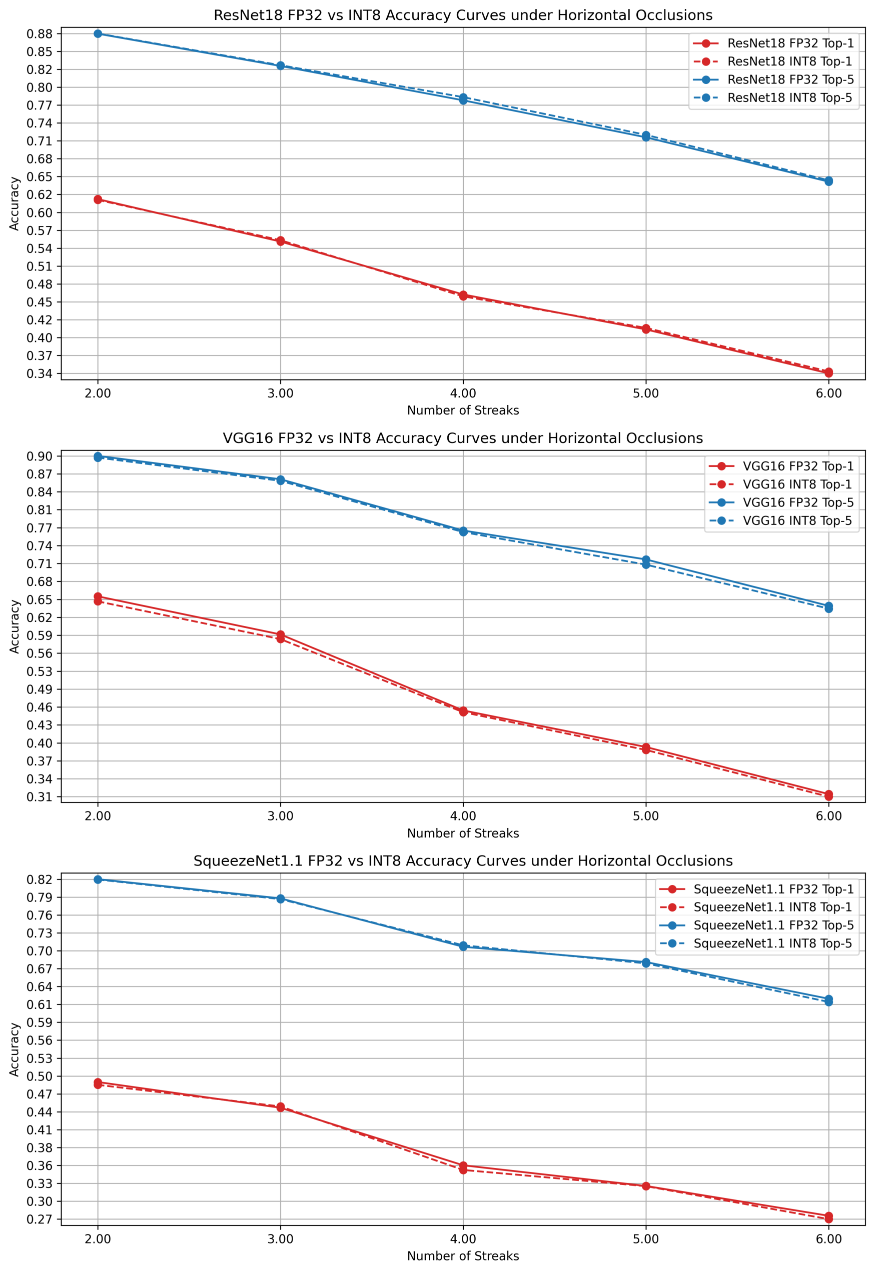


Figure 4.10: Top-1 and top-5 accuracies for each model's FP32/INT8 pair under various amounts of horizontal occlusion.

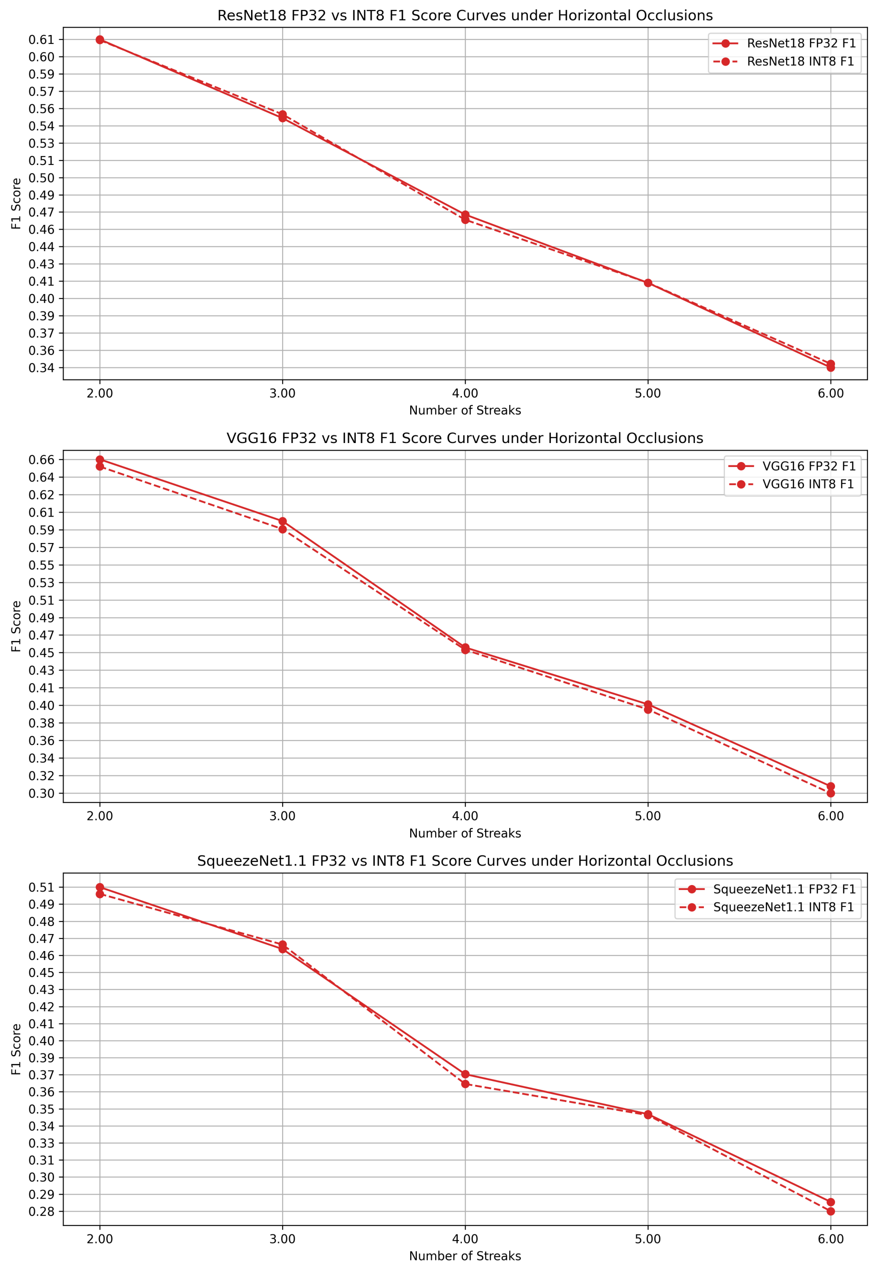


Figure 4.11: F1 scores for each model's FP32/INT8 pair under various amounts of horizontal occlusion.

Table 4.15: Tabulated top-1 accuracies under various amounts of horizontal occlusion.

Num. Streaks	SqueezeNet1_1		ResNet-18		VGG-16	
	FP32	INT8	FP32	INT8	FP32	INT8
2	0.4899	0.4854	0.6169	0.6157	0.6556	0.6469
3	0.4485	0.4506	0.5491	0.5512	0.5896	0.5815
4	0.3558	0.3483	0.4644	0.4614	0.4578	0.4548
5	0.3225	0.3222	0.4086	0.4110	0.3945	0.3894
6	0.2745	0.2691	0.3384	0.3414	0.3129	0.3084

Table 4.16: Tabulated top-5 accuracies under various amounts of horizontal occlusion.

	SqueezeNet1_1		ResNet-18		VGG-16	
Num. Streaks	FP32	INT8	FP32	INT8	FP32	INT8
2	0.8167	0.8161	0.8812	0.8815	0.8989	0.8959
3	0.7858	0.7846	0.8296	0.8308	0.8584	0.8557
4	0.7078	0.7102	0.7747	0.7798	0.7696	0.7669
5	0.6832	0.6811	0.7156	0.7195	0.7195	0.7105
6	0.6241	0.6190	0.6448	0.6469	0.6391	0.6343

Table 4.17: Tabulated F1 scores under various amounts of horizontal occlusion.

	SqueezeNet1_1		ResNet-18		VGG-16	
Num. Streaks	FP32	INT8	FP32	INT8	FP32	INT8
2	0.5060	0.5013	0.6141	0.6135	0.6623	0.6546
3	0.4624	0.4655	0.5492	0.5523	0.5954	0.5868
4	0.3741	0.3672	0.4696	0.4653	0.4585	0.4559
5	0.3459	0.3453	0.4133	0.4134	0.3968	0.3913
6	0.2841	0.2775	0.3435	0.3465	0.3080	0.3005

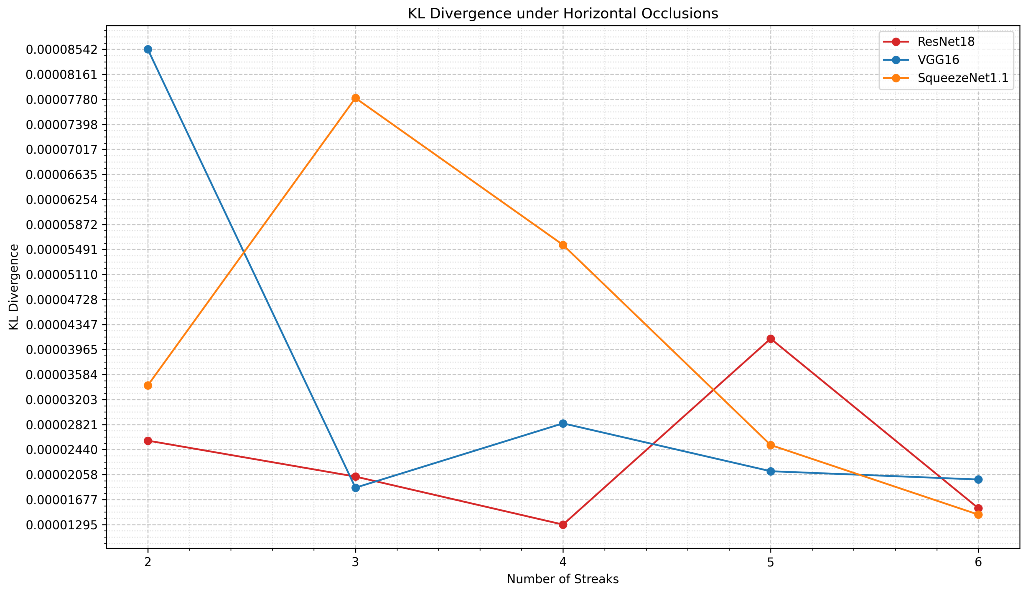


Figure 4.12: KL divergences for each model pair under various amounts of horizontal occlusion.

Table 4.18: KL divergences for each model pair under various amounts of horizontal occlusion.

Num. Streaks	SqueezeNet1_1	ResNet-18	VGG-16
2	3.4174e-05	2.5758e-05	8.5424e-05
3	7.7996e-05	2.0273e-05	1.8579e-05
4	5.5599e-05	1.2955e-05	2.8390e-05
5	2.5089e-05	4.1308e-05	2.1092e-05
6	1.4481e-05	1.5461e-05	1.9827e-05

In summary, our results elicit several clear trends. Firstly, we observe that AWGN's accuracy and F1 curves have somewhat negative exponential decay behavior and plateaus at $\sigma = 0.8$. Under Brownian noise, we see a more gradual descent in error with no plateau. Vertical occlusions yielded the highest degradation in all three model pairs, with an immediate drop off to sub 50% top-1 accuracy for each model, for only two streaks added. The effect of horizontal occlusions was less pronounced and had considerably higher performance metrics for the same number of streaks. Under all regimes, the INT8 model performs effectively, just as well as the FP32 model, with discrepancies typically being less than 1%. In terms of KL divergence, AWGN induced a peak divergence at $\sigma = 0.4$. Brownian noise yielded the highest overall divergences with maximums of 2.3271, 1.2215, and 1.6631 for SqueezeNet1_1, ResNet-18, and VGG-16, respectively, and had a steady near-linear increase with increasing σ. Under vertical and horizontal occlusions, we see KL divergences similar to the baseline results in that they are on the same order of magnitude or lower and near zero. In general, ResNet-18 exhibits the lowest KL divergence across all perturbation regimes, and VGG-16 and SqueezeNet1_1 the highest. Another interesting result is that in terms of accuracy and F1, VGG-16 outperforms both models under AWGN and Brownian noise, whereas ResNet-18 outperforms under highly structured regimes. Again, as discussed prior, this distinction may be indicative of each respective model's architecture - e.g., VGG-16's depth and sheer parameter count induce robustness under AWGN and Brownian noise, whereas ResNet-18's skip connections may mitigate degradation against complete highly structured perturbations like vertical and horizontal streaks.

Chapter 5

Conclusion

We have examined the robustness of FP32 and post-training INT8 quantized under input perturbations ranging from independent and identically distributed AWGN to spatially correlated Brownian noise to highly structured vertical and horizontal occlusions. We considered three SOTA models ranging in parameter count, including SqueezeNet1_1, ResNet-18, and VGG-16, and measured performance according to top-1 and top-5 accuracy and F1 score. To measure changes in class output confidence pre- and post-quantization, we employed KL. We implemented quantization according to the best practices detailed in [28], using PyTorch [32] paired with EasyQuant [36].

We have found that while degradation was observed amongst all models, substantial degradation in the quantized model *relative* to the full-precision model was not observed. Indeed, we see the highest degree of error between the models in VGG-16 under Brownian noise at $\sigma = 20$ with a top-1 accuracy drop of 1.62% and a 1.59% drop in F1. In terms of KL divergence, we see the most significant discrepancy in class output probabilities under the Brownian noise regime, followed by AWGN. Vertical and horizontal occlusions were on the same order of magnitude as the baseline results or near zero.

As stated in the introduction, this work sought to address deficiencies in the current body of research surrounding neural network quantization and, in doing so, de-risk their deployment in real-world scenarios and provide experimental data to understand their performance under various perturbation regimes. Based on these experimental findings, we can conclude that INT8 quantized networks do not exhibit ill-conditioning or exacerbated sensitivity under perturbation, relative to their FP32 counterpart, and, thus, are just as robust in these noise regimes. Moreover, these experiments suggest that degradation is a *macroscopic* phenomenon in that it is a result of overall model design - e.g., VGG's depth and parameter count trump

SqueezeNet1_1's lighter and reduced parameter space - rather than *parameter-to-parameter* precision. We have also identified a clear trade-off between traditional metrics like F1 and accuracy and model confidence or distribution similarity. For example, in the AWGN at σ, we see that VGG-16 outperforms ResNet-18 by a margin of 5% but also has a higher KL divergence. These experiments showed that bigger models like VGG-16 scoring higher top-1 accuracy do not necessarily make it ideal, especially given its significantly higher MACs, and GFLOPs. This is a crucial heuristic that could be used by designers in model development to identify candidate models based on expected deployment scenarios; for instance, if thermal noise is expected, one may consider AWGN, and if similarity in probabilities is more important than top-1 accuracy, ResNet-18, with 8.5x fewer GFLOPs than VGG-16, may be preferred. That is, we have identified a crucial trade-off between raw accuracy and F1 with KL divergence. We sought not to prove that, for example, VGG-16 is the most performant under perturbation, but rather, that there are trade offs when considering quantization in terms of accuracy and model similarity.

We propose several future research directions to further advance the understanding of neural network quantization and its robustness under perturbations. First, conducting this same study with quantization-aware training rather than post-training quantization may reveal subtle differences in robustness and further guide quantization scheme selection in real-world deployment. Additionally, exploring different quantization resolutions below 8-bit for both weights and activations could establish experimental bounds on the aggressiveness of quantization under perturbations. Another critical area of exploration is examining performance and robustness under adversarial perturbations, which differ fundamentally from random environmental perturbations as they can be optimized to degrade performance on a model-specific basis. Furthermore, expanding this study to include various model types and architectures and different tasks such as object localization and tracking, natural language processing, and regression tasks under perturbation may uncover opportunities for energy and hardware efficiencies in other machine learning applications.

Bibliography

[1] *Entropy, Relative Entropy, and Mutual Information*, chapter 2, pages 13–55. John Wiley Sons, Ltd, 2005.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

[3] Alan C. Bovik. *The Essential Guide to Image Processing.* Academic Press, 2009.

[4] Yen-Ching Chang and Jin-Tsong Jeng. Classifying images of two-dimensional fractional brownian motion through deep learning and its applications. *Applied Sciences*, 13(2), 2023.

[5] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016.

[6] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning, 2018.

[7] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. *Handwritten digit recognition with a back-propagation network*, page 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] Samuel Dodge and Lina Karam. Understanding how image quality affects deep neural networks, 2016.

[10] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: from adversarial to random noise, 2016.

[11] Jiong Gong, Haihao Shen, Guoming Zhang, Xiaoli Liu, Shane Li, Ge Jin, Niharika Maheshwari, Evarist Fomenko, and Eden Segal. Highly efficient 8-bit low precision inference of convolutional neural networks with intelcaffe, 2018.

[12] Issam Hammad and Kamal El-Sankary. Impact of approximate multipliers on vgg deep learning network. *IEEE Access*, 6:60438–60444, 2018.

[13] Issam Hammad and Kamal El-Sankary. Practical considerations for accuracy evaluation in sensor-based machine learning and deep learning. *Sensors*, 19(16), 2019.

[14] Issam Hammad, Kamal El-Sankary, and Jason Gu. Deep learning training with simulated approximate multipliers. *CoRR*, abs/2001.00060, 2020.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[16] Xiangteng He and Yuxin Peng. Fine-grained visual-textual representation learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(2):520–531, February 2020.

[17] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

[18] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[22] Fengfu Li, Bin Liu, Xiaoxing Wang, Bo Zhang, and Junchi Yan. Ternary weight networks, 2022.

[23] Ling Li, Issam Hammad, and Kamal El-Sankary. Dual segmentation approximate multiplier. *Electronics Letters*, 57, 05 2021.

[24] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks, 2016.

[25] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[26] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[27] J. Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.

[28] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *CoRR*, abs/2106.08295, 2021.

[29] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008.

[30] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study, 2018.

[31] Xinwei Ou, Zhangxin Chen, Ce Zhu, and Yipeng Liu. Low rank optimization for efficient deep learning: Making a balance between compact architecture and fast training, 2023.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Hong Qian. *Fractional Brownian Motion and Fractional Gaussian Noise*, pages 22–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[35] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016.

[36] Oscar Savolainen. Oscarsavolainendr/easyquant, Jun 2024.

[37] Jaydip Sen, Abhiraj Sen, and Ananda Chatterjee. Adversarial attacks on image classification models: Analysis and defense, 2023.

[38] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[40] Olivia Weng. Neural network quantization for efficient inference: A survey, 2023.

[41] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[42] Zhongzhi Yu, Yemin Shi, Tiejun Huang, and Yizhou Yu. Kernel quantization for efficient network compression, 2020.

Appendix A

Quantization Python Code

```
import torch
import torchvision.models as models
import torch.quantization as tq
from torch.ao.quantization.quantize_fx import prepare_qat_fx
from torch.ao.quantization.qconfig_mapping import QConfigMapping
from quant_modules.state_toggling import enable_fake_quant,
↪  enable_PTQ_observer
from quant_modules.learnable_fake_quantize import EQLearnableFakeQuantize

def load_models(num_classes=70, resnet18_path=None, vgg16_path=None,
↪  squeezenet1_1_path=None):
    # Fetch models from PyTorch
    resnet18 = torchvision.models.resnet18(weights=None)
    vgg16 = torchvision.models.vgg16(weights=None)
    squeezenet1_1 = torchvision.models.squeezenet1_1(weights=None)
    # Update models to handle 70 classes
    resnet18.fc = nn.Linear(resnet18.fc.in_features, num_classes)
    vgg16.classifier[6] = nn.Linear(vgg16.classifier[6].in_features,
    ↪  num_classes)
    squeezenet1_1.classifier[1] = nn.Conv2d(512, num_classes,
    ↪  kernel_size=(1,1), stride=(1,1))
    squeezenet1_1.num_classes = num_classes
    # Load pre-trained parameters from disk
    resnet18.load_state_dict(torch.load(resnet18_path, map_location="cpu"))
    vgg16.load_state_dict(torch.load(vgg16_path, map_location="cpu"))
    squeezenet1_1.load_state_dict(torch.load(squeezenet1_1_path,
    ↪  map_location="cpu"))
    return resnet18, vgg16, squeezenet1_1
```

```
def quantize(model):
    # Set weight quantization parameters. Initialize scale and zero_point
    learnable_weight = lambda channels: EQLearnableFakeQuantize.with_args(
        observer=tq.PerChannelMinMaxObserver, # Set histogram type
        quant_min=-128, # Symmetric 8-bit min.
        quant_max=127, # Sypemtric 8-bit max.
        dtype=torch.qint8, # Set datatype to int8
        qscheme=torch.per_channel_symmetric, # Specify resolution
        scale=0.1, # Initial scale
        zero_point=0.0, # Initial zero point
        use_grad_scaling=True, # Scale gradients
        channel_len=channels, # Length of channel dimension
    )
    # Set activation quantization parameters. Initialize scale and
    ↪   zero_point
    learnable_act = EQLearnableFakeQuantize.with_args(
        observer=tq.HistogramObserver, # Set histogram type
        quant_min=0, # Affine 8-bit min.
        quant_max=255, # Affine 8-bit max.
        dtype=torch.quint8, # Set datatype to uint8
        qscheme=torch.per_tensor_affine, # Specify resolution
        scale=0.1, # Initial scale value
        zero_point=0.0, # Initial zero point
        use_grad_scaling=True, # Use gradient scaling
    )
    torch.backends.quantized.engine = "qnnpack" # Set quantization engine
    ↪   to qnnpack
    qconfig_mapping = QConfigMapping() # Create qconfig instance
    # Assign qconfigs to each module in model
    for name, module in model.named_modules():
        if hasattr(module, "out_channels"):
            qconfig = tq.QConfig(
                activation=learnable_act,
                weight=learnable_weight(channels=module.out_channels),
            )
            qconfig_mapping.set_module_name(name, qconfig)
            module.qconfig = qconfig
    example_inputs = (torch.randn(1, 3, 224, 224),) # Random input for
     ↪  calibration
    model.eval() # Set model to evaluation mode
    quant_model = prepare_qat_fx(model, qconfig_mapping, example_inputs) #
     ↪  Prepare and calibrate quantized model
    quant_model.eval() # Set quant model to evaluation mode
    # Enable quantization on model
    quant_model.apply(enable_fake_quant)
    quant_model.apply(enable_PTQ_observer)
    return quant_model # Return quantized model
```

Appendix B

Average KL Divergence Python Code

```python
import numpy as np
from scipy.stats import entropy

def compute_kl_divs(p_fp32, p_int8, m):
    """
    Compute kl divergence of int8 probabilities (p_int8) to fp32
    ↪  probabilities (p_fp32)
    for each x (perturbation level) value in m (list of perturbation levels).
    """
    kl_divs = []
    for i, x in enumerate(m):
        p_fp32_sigma = p_fp32[i]  # List of 3333 images with 70
        ↪  probabilitiess each
        p_int8_sigma = p_int8[i]  # List of 3333 images with 70 probilities
        ↪  each
        kl_div_sum = 0
        for j in range(len(p_fp32)):
            p_fp32_sigma_image = p_fp32_sigma[j]  # Each individual set of 70
            ↪  probabilities for a given image
            p_int8_sigma_image = p_int8_sigma[j]  # Each individual set of 70
            ↪  probabilities for a given image
            kl_div_image = entropy(p_fp32_sigma_image, p_int8_sigma_image) #
            ↪  Compute KL divergence for this image
            kl_div_sum += kl_div_image # Accumulate sum

        avg_kl_div = kl_div_sum / 3333 # Compute average KL divergence for
        ↪  this x value
        kl_divs.append((x, avg_kl_div)) # Append (x, avg_kl_div) to the list
    return kl_divs # Return the list of (x, avg_kl_div) pairs
```