ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Adversarial Robustness Strategies for Neural Networks in Hardware

Διπλωματική Εργασία

Αλέξανδρος Πατεράκης

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Adversarial Robustness Strategies for Neural Networks in Hardware

Διπλωματική Εργασία

Αλέξανδρος Πατεράκης

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε απο την τριμελή εξεταστική επιτροπή στις 2 Απριλίου 2024.

....	....	....
Δημήτριος Σούντρης	**Παναγιώτης Τσανάκας**	**Σωτήριος Ξύδης**
(Καθηγητής Ε.Μ.Π.)	(Καθηγητής Ε.Μ.Π.)	(Επίκουρος Καθηγητής Ε.Μ.Π.)

Αθήνα, Απρίλιος 2024

(Υπογραφή)

Αλέξανδρος Πατεράκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών, Ε.Μ.Π.

Περίληψη

Τα νευρωνικά δίκτυα ερευνώνται εδώ και αρκετά χρόνια, ωστόσο έχουν παρουσιάσει σημαντική άνοδο στην δημοσιότητα μόνο την τελευταία δεκαετία. Η άνοδος αυτή έχει συμβάλει στην ραγδαία ανάπτυξη διαφόρων τομέων όπως την ιατρική, την γεωργία, τον αυτοματισμό αυτοκινήτων και πολλούς άλλους. Παράλληλα, οι δυνατότητες των δικτύων στην ανίχνευση αντικειμένων, αναγνώριση εικόνων, επεξεργασία φυσικής γλώσσας και λήψης απόφασης έχουν οδηγήσει σε πολλαπλές ανακαλύψεις με εξαιρετικά επίπεδα ακριβείας και αποτελεσματικότητας. Παρά την συνεχή ανοδική πορεία τους, η ασφάλεια και η ανθεκτικότητα, χαρακτηριστικά που αποτελούν βασικούς συντελεστές των νευρωνικών δικτύων, παραμελούνται. Παρόλο που συνηθίζουμε στην χρήση νευρωνικών δικτύων όλο και περισσότερο στην καθημερινή μας ζωή, η ευαισθησία τους σε στοχευμένες εναντίων τους επιθέσεις, δημιουργεί ανησυχίες σχετικά με την αξιοπιστία και την ασφάλειά τους. Επιτηδευμένες επιθέσεις όπως το HopSkipJump, οι οποίες εκμεταλλεύονται τα ευπαθή σημεία των μοντέλων προσθέτοντας μικρές ανεπαίσθητες διαταραχές στα δεδομένα, αποτελούν σοβαρό ελάττωμα σε κρίσιμες ως προς την ασφάλεια εφαρμογές όπως τα αυτόνομα οχήματα.

Η παρούσα διπλωματική, ασχολείται με το κρίσιμο αυτό ζήτημα, διερευνώντας στρατηγικές ενίσχυσης της ανθεκτικότητας των νευρωνικών δικτύων αναγνώρισης εικόνων ενάντια στις συγκεκριμένες επιθέσεις. Επιπλέον, εξετάζεται ο αντίκτυπος της εγκατάστασης των δικτύων σε συσκευές περιορισμένων πόρων, ως προς την απόδοση τους, διερευνώντας έτσι τρόπους με τους οποίους συσκευές, όπως η πλατφόρμα Versal, μπορούν να αξιοποιηθούν για την ενίσχυση της ανθεκτικότητας σε περιβάλλοντα Edge (Edge Computing) προσομοιώνοντας ρεαλιστικές εφαρμογές. Συγκεκριμένα, στην εργασία αυτή, επικεντρωνόμαστε στις δημοφιλείς αρχιτεκτονικές ResNet20, ResNet56 και MobileNetV2. Χρησιμοποιώντας τα CIFAR-10 και FashionMNIST datasets, εφαρμόζουμε στα μοντέλα μια σειρά από τεχνικές, όπως την προ-επεξεργασία (Preprocessing), την κβάντιση (Quantization) και την επανεκπαίδευση με δεδομένα τα αποτελέσματα των επιθέσεων Adversarial Retraining. Τέλος, ο αντίκτυπος των μηχανισμών αυτών ποσοτικοποιείται με χρήση μετρικών PSNR, αναδεικνύοντας σημαντικές βελτιώσεις που κυμαίνονται μεταξύ 37% και 49%, συμβάλλοντας έτσι στην ανάπτυξη ασφαλέστερων και πιο αξιόπιστων νευρωνικών δικτύων.

Λέξεις Κλειδιά: Νευρωνικά Δίκτυα, Αναγνώριση Εικόνων, Hardware, Versal, Edge Computing, Προεπεξεργασία, Επιτηδευμένη Επίθεση, Adversarial Retraining

Abstract

While Neural Networks have been in research for multiple years, they have experienced a significant surge in popularity only in the last decade. This surge has helped revolutionize various sectors and fields from medical to agriculture to automotive automation and many more, while their capabilities on object detection, computer vision, natural language processing and decision-making have led to multiple breakthroughs with exceptional levels of accuracy and efficiency. However, despite their exponential growth, their security and robustness, which are key factors of Neural Networks, seem to be overlooked. Even though we are becoming more accustomed to using Neural Networks in our everyday lives, their susceptibility to adversarial attacks can create concerns over their reliability and safety. Adversarial attacks, such as HopSkipJump, which exploit vulnerabilities in models by introducing small imperceptible perturbations in the input data, can be a major flaw in safety-critical applications such as autonomous vehicles.

This thesis, addresses the crucial issue by investigating strategies to enhance the robustness of image classification networks against adversarial attacks. Additionally, it considers the impact of hardware deployment on a Neural Network's performance, exploring how hardware resources, including the Versal Platform, can be leveraged to enhance the robustness of models in edge computing environments and emulate real-world deployment. Specifically, this work focuses on popular lightweight model architectures, namely ResNet20, ResNet56 and MobileNetV2. By employing CIFAR-10 and FashionMNIST Datasets, a series of techniques, including adversarial retraining, preprocessing and quantization are applied on these models to improve their robustness. Furthermore, the impact of such mechanisms is quantified using PSNR metrics, demonstrating significant improvements in robustness, with enhancements ranging from 37% to 49% thus contributing to the development of more secure and reliable Neural Networks for practical applications.

Keywords: Neural Networks, Image classification, Hardware, Versal, Edge Computing, Preprocessing, Adversarial Attack, Adversarial Retraining

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου, κ.Δημήτριο Σούντρη, για την επίβλεψη της διπλωματικής μου, αλλα επίσης για την επιστοσύνη και την μοναδική ευκαιρία που μου παρείχε για την εκπόνησή της. Εν συνεχεία, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα Δημήτριο Δανόπουλο για την υποστήριξη, καθοδήγηση και βοήθεια που μου προσέφερε. Επίσης θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Μικροϋπολογιστών και Ψηφιακών Συστημάτων για το χαρμόσυνο και φιλόξενο κλίμα που μου προσέφεραν. Τέλος, ευχαριστώ τους φίλους και την οικογένειά μου για όλα όσα μου έχουν προσφέρει όλα αυτά τα χρόνια και για την συμπαράσταση και αγάπη που μου έχουν δώσει.

Αλέξανδρος Πατεράκης
Απρίλιος, 2024

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
DNN	Deep Neural Network
FNN	Feed-Forward Neural Network
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory Neural Network
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
QAT	Quantization Aware Training
PTQ	Post-Training Quantization
ResNet	Residual Network
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
DSP	Digital Signal Processing
CLB	Configurable Logic Block
SoC	System-on-Chip
NoC	Network-on-Chip
SIMD	Single Instruction Multiple Data
VLIW	Very Large Instruction Word

Chapter 1

Εκτεταμένη Ελληνική Περίληψη

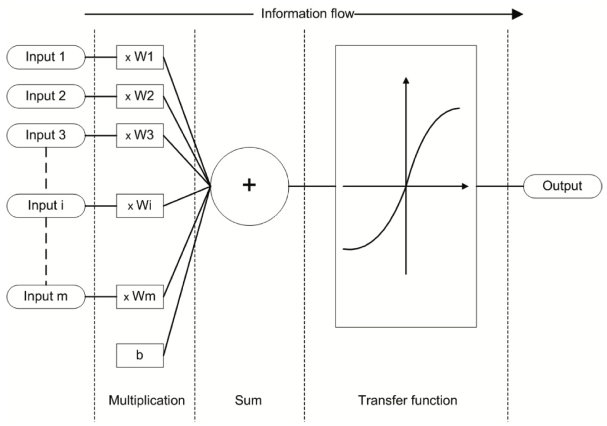
1.1 Εισαγωγή

Η Τεχνητή Νοημοσύνη (TN) έχει έρθει στο επίκεντρο της προσοχής τα τελευταία χρόνια, επηρεάζοντας διάφορες πτυχές της ζωής μας. Απο τον ιατρικό τομέα μεχρι και την αυτοκινητοβιομηχανία, αυτές οι τεχνολογίες καθοδηγούμενες απο βαθιά νευρωνικά δίκτυα (DNN) ενσωματώνονται όλο και περισσότερο στην καθημερινότητά μας. Ωστόσο, αυτή η ενσωμάτωση αποκαλύπτει μια κρίσιμη ευπάθεια των νευρωνικών δικτύων: την ευαισθησία σε επιτηδευμένες επιθέσεις όπως την HopSkipJumpAttack. Τέτοιου είδους επιθέσεις αφορούν κυρίως νευρωνικά δίκτυα αναγνώρισης εικόνων, όπου εισάγοντας μικρες ανεπαίσθητες διαταραχές στα δεδομένα εισόδου, προκαλούν λανθασμένες εκτιμήσεις. Παρά την αποτελεσματικότητα των DNN, η ευαισθησία σε αυτές τις επιθέσεις εγείρει ανησυχίες για την αξιοπιστία τους καθιστώντας αναγκαία την έρευνα για τρόπους προστασίας. Στην παρούσα διπλωματική εργασία, διερευνούμε στρατηγικές για την ενίσχυση της ανθεκτικότητας νευρωνικών δικτύων, εφαρμοζόμενα σε συσκευές περιορισμένων πόρων, κατα αυτων των επιτηδευμένων επιθέσεων. Αναλύουμε τα ResNet20, ResNet56 και MobileNetV2 στα σύνολα δεδομένων CIFAR-10 και FashionMNIST. Επιπλέον, διερευνούμε την επίδραση ειδικού σκοπού συσκευών στα νευρωνικά δίκτυα αυτά με τη χρήση της πλατφόρμας Versal και του συνόλου εργαλείων VitisAI Toolset. Στόχος μας είναι να γεφυρώσουμε θεωρητικές εξελίξεις στον τομέα προστασίας των δικτύων με εφαρμογές στον πραγματικό κόσμο, αντιμετωπίζοντας αποτελεσματικά τις ευαισθησίες των νευρωνικών δικτύων.

1.2 Θεωρητικό Υπόβαθρο

Τεχνητά Νευρωνικά Δίκτυα

Τα Τεχνητά Νευρωνικά Δίκτυα, θεμελιώδη στην τεχνητή νοημοσύνη, μιμούνται την συμπεριφορά και μορφή των βιολογικών νευρωνικών δικτύων. Αντίστοιχα με τα βιολογικά τους ομόλογα, τα τεχνητά δίκτυα βασίζονται σε πολλαπλούς νευρώνες για την λειτουργία τους [12]. Κάθε τεχνητος νευρώνας, υπεύθυνος για διαφορετικές λειτουργίες, σταθμίζει τα δεδομένα εισόδου πριν αθροιστούν και μεταφερθούν σε μια συνάρτηση ενεργοποίησης. Αυτή η διαδικασία, που εντοπίζεται στην εικόνα 1.1 και αναπαρίσταται στην εξίσωση (1.1), αποτελεί την κύρια βάση για όλα τα νευρωνικά δίκτυα, επιτρέποντας την μάθηση και προσαρμογή στα αντίστοιχα δεδομένα. Επιπλέον, χρησιμοποιώντας πολλαπλές τεχνικές ειναι δυνατή η κατασκευή σύνθετων νευρωνικών δικτύων σχεδιασμένα για συγκεκριμένες εφαρμογές και συσκευές.



Σχήμα 1.1: Αναπαράσταση τεχνητού νευρώνα, Πηγή [12]

$$y(k) = F\left(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b\right) \tag{1.1}$$

Ανάλογα την αντίστοιχη αρχιτεκτονική, πλήθος τεχνητών νευρώνων διατάσσεται σε επίπεδα. Εισάγοντας αυτα τα επίπεδα μεταξύ των σταδίων εισόδου και εξόδου, δημιουργείται ενα βασικό νευρωνικό δίκτυο ενω, με την προσθήκη πολλαπλών επιπέδων σταδιακά παράγονται πιο σύνθετες αρχιτεκτονικές. Για την βελτιστοποίηση και καλύτερη προσαρμογή των δικτύων στην κάθε εφαρμογή, μπορούμε επίσης να προσαρμόσουμε τα χαρακτηριστικά των επιπέδων, απο τις συναρτήσεις ενεργοποίησης των νευρώνων μέχρι και το μέγεθος τους. Οι επιλογές μας για συναρτήσεις ενεργοποίησης έχουν ως εξής:

- Βηματική συνάρτηση, όπου αν η είσοδος περάσει ενα προκαθορισμένο όριο, η έξοδος γίνεται 1 όπως βλέπουμε στην εξίσωση (1.2).

$$y_1 = \begin{cases} 1, & \text{ιφ } x_1 \cdot w_1 \geq threshold \\ 0, & \text{ιφ } x_1 \cdot w_1 < threshold \end{cases} \tag{1.2}$$

- Γραμμική συνάρτηση, όπου η είσοδος περνάει στην έξοδο όπως παρατηρούμε στην εξίσωση (1.3).

$$F(x) = x \tag{1.3}$$

- Σιγμοειδής Μη Γραμμική συνάρτηση, όπου παράγει τιμές εντός του [0,1] πεδίου με μη γραμμικό τρόπο εκφραζόμενη απο την εξίσωση (1.4).

$$\sigma(x) = \frac{1}{1+e^{-x}}, \ \ F(x) \in [0,1] \tag{1.4}$$

- Υπερβολική συνάρτηση, όπου παράγει τιμές μεταξύ [-1,1] (1.5) και εντοπίζεται συνήθως σε ενδιάμεσα κρυφά επίπεδα σύνθετων δικτύων.

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \ \ F(x) \in [-1,1] \tag{1.5}$$

- Rectified Linear Unit (ReLU)[28], χαρακτηριζόμενη απο (1.6) και ειναι απο τις πιο συνηθισμένες συναρτήσεις.

$$f(x) = max(0,x) \tag{1.6}$$

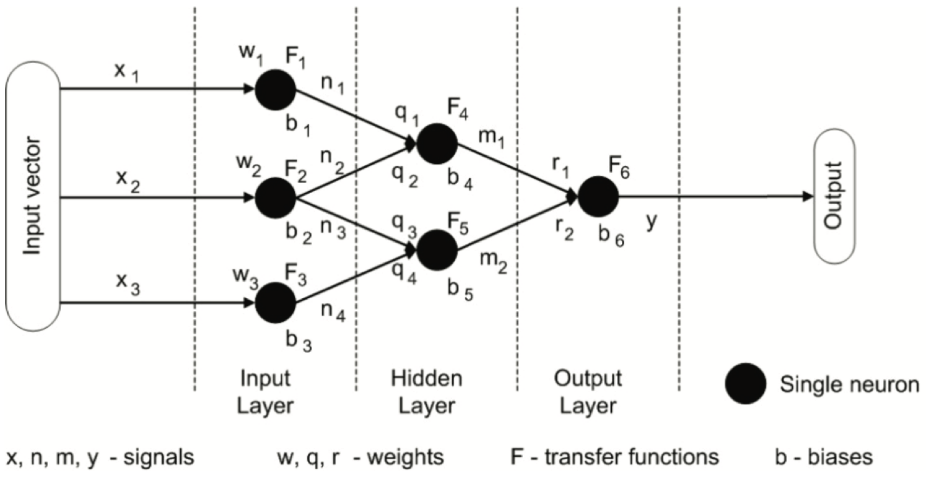
- Softmax, η οποία εντοπίζεται στις εξόδους μοντέλων ανίχνευσης πολλαπλων κλάσεων και εκφράζεται ως (1.7).

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1.7}$$

Με την χρήση πολλαπλών συναρτήσεων αλλα και επιπέδων, μπορούν να σχεδιαστούν διάφορες αρχιτεκτονικές βαθιών νευρωνικών δικτύων. Ακολουθούν οι πιο γνωστοί τύποι δικτύων:

- Νευρωνικά Δίκτυα Προώθησης

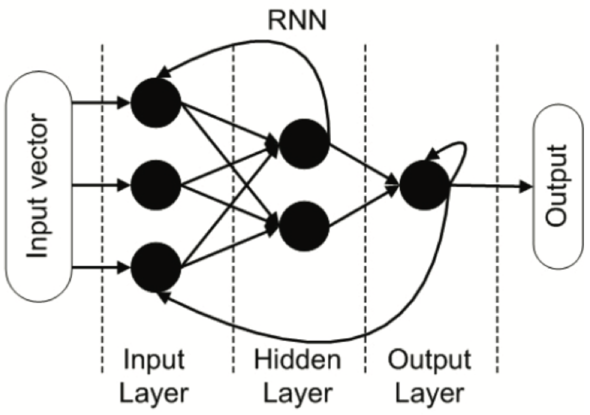
  Τα δίκτυα αυτά αποτελούν απο την πιο βασική αρχιτεκτονική σύνθετων δικτύων. Η χαρακτηριστική τους λειτουργία ειναι η δυνατότητα μονόδρομης ροής πληροφοριών απο την είσοδο στην έξοδο περιορίζοντας κάθε διασύνδεση με προηγούμενα επίπεδα ή νευρώνες.



Σχήμα 1.2: Παράδειγμα Προωθητικού δικτύου 2 επιπέδων, Πηγή: [12]

- Αναδρομικά Νευρωνικά Δίκτυα

  Σε αντίθεση με τα δίκτυα προώθησης, τα αναδρομικά βασίζονται στον διαμοιρασμό πληροφοριών μεταξύ επιπέδων και νευρώνων είτε με την αναδρομή ίδιων δεδομένων στους νευρώνες είτε με την μεταφορά πληροφοριών μεταξύ επιπέδων.



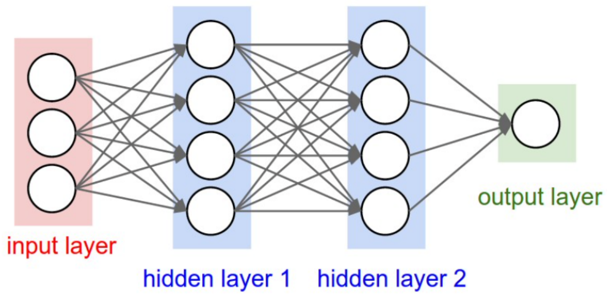
Σχήμα 1.3: Παράδειγμα Αναδρομικού δικτύου 2 επιπέδων, Πηγή: [12]

- Παραγωγικά Αντιπαλικά Δίκτυα

  Αποτελώντας δική τους κλάση στον τομέα των νευρωνικών δικτύων, αποτελούνται απο δύο διαφορετικά μοντέλα. Το πρώτο παράγει συνεχώς δεδομένα στον τύπο που έχει προκαθοριστεί ενω το δεύτερο τα αξιολογεί [32].

- Συνελικτικά Νευρωνικά Δίκτυα

  Συνήθως χρησιμοποιούνται σε εφαρμογές αναγνώρισης μοτίβων και εικόνων. Τα δίκτυα αυτά ξεχωρίζουν για την δυνατότητα τους να διαχειρίζονται περίπλοκα και πολυπληθή δεδομένα με δραστικά μειωμένες παραμέτρους συγκριτικά με τυπικά βαθιά νευρωνικά δίκτυα [35]. Θα αναλύσουμε την αρχιτεκτονική τους σε επόμενο στάδιο εντός αυτής της ενότητας.



Σχήμα 1.4: Παράδειγμα Συνελικτικού Νευρωνικού Δικτύου, Πηγή: [19]

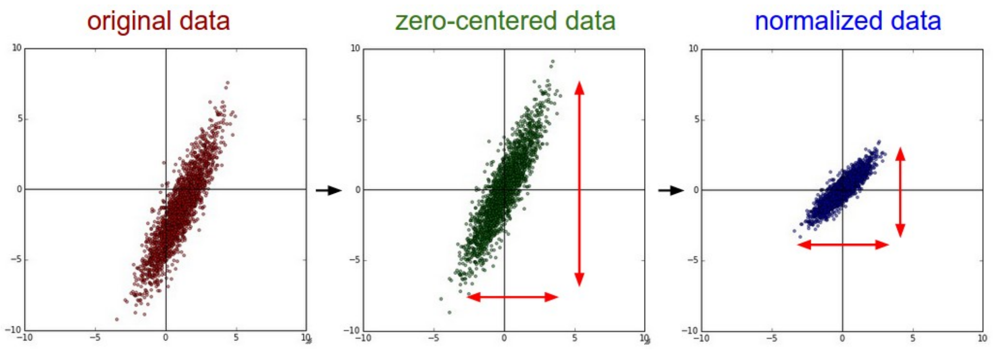
Εκπαίδευση

Κατα την εκπαίδευση των νευρωνικών δικτύων εφαρμόζονται πολλαπλές τεχνικές και μέθοδοι για την ενίσχυση της ακρίβειάς τους.

1. **Προ-επεξεργασία δεδομένων**

   Πριν την εκκίνηση της εκπαίδευσης ενος δικτύου με τα επιλεγμένα δεδομένα, είναι σημαντικό να διασφαλίσουμε την ομοιόμορφη διαβάθμισή τους σε όλες τις διαστάσεις τους. Για την λειτουργία αυτή επιλέγουμε την προ-επεξεργασία των δεδομένων σε δυο στάδια. Αρχικά, εκτελείται **αφαίρεση μέσου όρου** κατα την οποία αφαιρώντας τον μέσο όρο απο την κάθε διάσταση των δεδομένων, επικεντρώνονται στο 0. Το δεύτερο στάδιο αποτελείται απο την **κανονικοποίηση** των δεδομένων κατα την οποία διαιρείται η μέση απόκλιση της κάθε διάστασης απο αυτή. Με τον τρόπο αυτό, τα δεδομένα διατηρούν ομοιόμορφες τιμές κεντραρισμένες στο 0. Οι εξισώσεις (1.8) χαρακτηρίζουν τα δυο αυτά στάδια αλλά και την σύνθεσή τους ενώ το η διαδικασία απεικονίζεται στην 1.5.

$$\begin{gathered}\text{Θεωρώντας πίνακα } x \text{ τα δεδομένα } d \text{ διαστάσεων}\\ x[i] = x[i] - mean[i]\\ x[i] = \frac{x[i]}{std[i]}\\ x[i] = \frac{x[i] - norm[i]}{std[i]}\end{gathered} \tag{1.8}$$



Σχήμα 1.5: Διαδικασία κανονικοποίησης, Πηγή: [19]

2. **Υπερεκπαίδευση**

   Η υπερεκπαίδευση ειναι ενα ακόμα σημαντικό κομμάτι το οποίο εύκολα οδηγει σε πτώσεις απόδοσης ενος δικτύου. Ορίζεται ως η ταση ενος νευρωνικού δικτύου να προσαρμόζεται στα δεδομένα εκπαίδευσης αποτυγχάνοντας να μαθαίνει μοτίβα και υποκείμενες σχέσεις. Για την αποφυγή αυτού του φαινομένου εφαρμόζονται πολλαπλές τεχνικές όπως κανονικοποίηση παραμέτρων και επικύρωση ακρίβειας.

   Αρχικα, με την κανονικοποίηση παραμέτρων εισάγονται ποινές στις παραμέτρους του μοντέλου. Οι ποινές αυτές, ανάλογες είτε της απόλυτης τιμής των παραμέτρων είτε του τετραγώνου τους, τείνουν να μηδενίζουν τις παραμέτρους προωθώντας την σπανιότητα των δεδομένων και εν τέλη βελτιώνοντας την γενίκευση του μοντέλου.

   Η επικύρωση της ακρίβειας συνήθως συνδυάζεται με πρόωρο τερματισμό. Με αυτή την τεχνική, κατα την εκπαίδευση του μοντέλου, χρησιμοποιείται ενα μικρό κομμάτι των δεδομένων ως επικύρωση, με το οποίο γίνεται εκτίμηση της απόδοσης του μοντέλου. Επομένως, μόλις η εκτίμηση αυτή φτάσει τα επιθυμητά ποσοστά η εκπαίδευση σταματά.

Ενα ακομα αναπόσπαστο κομμάτι των νευρωνικων ειναι οι βελτιστοποιητές. Πρόκειται για μια κατηγορία αλγορίθμων υπεύθυνοι για την βελτιστοποίηση του νευρωνικού δικτύου και την ελαχιστοποίηση της συνάρτησης απώλειας κατα την διάρκεια της εκπαίδευσης ενημερώνοντας τις παραμέτρους.

Αρχικά, η συνάρτηση απώλειας ποσοτικοποιεί τις διαφορές μεταξύ εξόδων του δικτύου και τις σταθερές αλήθειες απο τα σύνολα των δεδομένων. Επιπλέον, οι αλγόριθμοι αυτοί ανάλογα την υλοποίησή τους φέρουν διάφορες σημαντικές παραμέτρους όπως:

- Ρυθμός εκμάθησης, με τον οποίο ρυθμίζεται η ταχύτητα με την οποία μαθαίνει ενα δίκτυο στα δεδομένα.
- Μέγεθος παρτίδας, όπου προκαθορίζει το ποσό των δεδομένων που χρησιμοποιούνται σε καθε επανάληψη της εκπαίδευσης.

- Φθορά βαρών, με την οποία ορίζεται ο ρυθμός φθοράς των παραμέτρων του δικτύου.

Κβάντωση

Η Κβάντωση των νευρωνικών δικτύων ειναι ενα απαραίτητο στάδιο για την μεταφορά σε ειδικού σκοπού συσκευές, FPGA και μικροελεγκτές. Τα νευρωνικά δίκτυα σχεδιάζονται κατα κύριο λόγο με αριθμητική κινητής υποδιαστολής αξιοποιώντας την υψηλή ακρίβειά της. Ωστόσο, οι συσκευές αυτές, δεν υποστηρίζουν αυτούς τους τύπους δεδομένων απαιτώντας μετατροπή του δικτύου σε ακέραιου τύπου αριθμητική. Τον σκοπό αυτό πληρεί η Κβάντωση μετατρέποντας της παραμέτρους και τις συναρτήσεις ενεργοποίησης του δικτύου. Εφαρμοζόμενη υπο πολλές μορφές, η Κβάντωση εκφράζεται στην εξίσωση (1.9) [25].

$$Q(r) = Int(r/S) - Z$$

Όπου r είναι η πραγματική τιμή εισόδου, S ο παράγοντας προσαρμογής και Z ενα ακέραιο σημείο μηδέν. (1.9)

Ο παράγοντας προσαρμογής S χαρακτηρίζεται ως εξής (1.10).

$$S = \frac{\beta - \alpha}{2^{\beta} - 1} \tag{1.10}$$

Το ευρος [α,β] ορίζει το πεδιο προσαρμογής ενω η επιλογή του κατηγοριοποιεί την Κβάντωση σε συμμετρική ή ασύμμετρη. Επιπλέον κατα την συμμετρική Κβάντωση, ο παράγοντας S χωρίζεται σε πλήρους και περιορισμένου εύρους Κβάντωση όπως παρατηρούμε στην (1.11). Θεωρώντας Κβάντωση 8-bit, το πλήρες εύρος ορίζεται ώς [-128,127] ενω το περιορισμένο [-127,127].

$$S = \frac{2max(|r|)}{2^{n} - 1}$$

Πλήρες εύρος Κβάντωσης

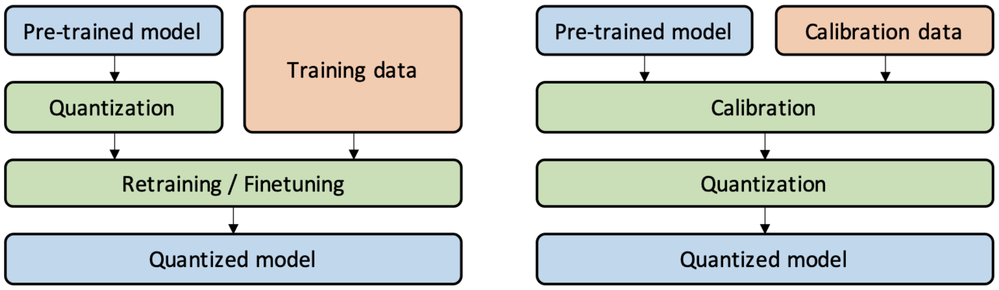
$$S = \frac{max(|r|)}{2^{n-1} - 1} \tag{1.11}$$

Περιορισμένο εύρος

Αξίζει να σημειωθεί οτι συνήθως προτιμάται η πλήρους εύρους συμμετρική Κβάντωση για λόγους απλότητας και απόδοσης.

Η Κβάντωση των δικτύων εφαρμόζεται με δυο τρόπους. Αρχικά, μεσω Εκπαίδευσης με επίγνωση κβαντισμού ειναι δυνατή η διατήρηση της ακρίβειας και κατάστασης του μοντέλου. Ωστόσο, αυτη η διαδικασία κβαντίζει τις παραμέτρους κατα την διάρκεια της εκπαίδευσης μετά τα περάσματα των βελτιστοποιητών. Για τον λόγο αυτό, η χρήση της σε ειδικού σκοπού συσκευές ειναι αδύνατη.

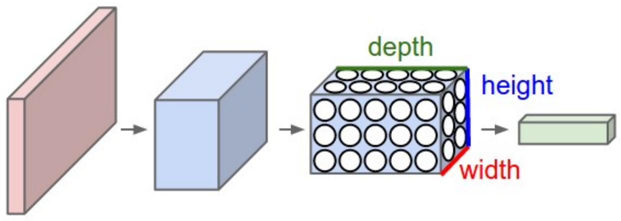
Αντίθετα, ο κβαντισμός μετά την εκπαίδευση, μετατρέπει όλες τις συναρτήσεις ενεργοποίησης και παραμέτρους του μοντέλου χωρις ενδιάμεσα στάδια. Αυτός ο τρόπος Κβάντωσης ωστόσο επιφέρει πτώσεις στην ακρίβεια του μοντέλου και χρειάζεται ενα μέρος των δεδομένων για βαθμονόμιση των παραμέτρων.



Σχήμα 1.6: Εκπαίδευση με επίγνωση Κβάντωσης (Αριστερά) - Κβάντωση μετά την εκπαίδευση (Δεξια), Πηγή: [25]

Συνελικτικά Νευρωνικά Δίκτυα

Τα συνελικτικά νευρωνικά δίκτυα ειναι μια ειδική κατηγορία νευρωνικών δικτύων προσαρμοσμένα για διαχείριση εικόνων με χρήση συνελικτικών πράξεων. Σε αντίθεση με τα τυπικά πλήρη συνδεδεμένα νευρωνικά δίκτυα, τα συνελικτικά δίκτυα διατάσσουν τους νευρώνες με αντίστοιχο τρισδιάστατο τρόπο όπως η δομή των εικόνων χωρίς να διατηρούν όλες τις συνδέσεις μεταξύ τους.



Σχήμα 1.7: Παράδειγμα Συνελικτικού Δικτύου 3ων επιπέδων, Πηγή: [19]

Τα δίκτυα αυτά αποτελούνται απο τρεις διαφορετικούς τύπους επιπέδων: το συνελικτικό επίπεδο, το συγκεντρωτικό επίπεδο και ενα πλήρες συνδεδεμένο.

Το συνελικτικό επίπεδο, συνδυαζόμενο συνήθως με ενα ReLU επίπεδο, αποτελείται απο ενα σύνολο φίλτρων που σαρώνει την είσοδο και ανανεώνει τα βάρη του αντιστοίχως κατα την εκπαίδευση. Με τον τρόπο αυτό το κάθε φίλτρο ευαισθητοποιείται σε διαφορετικά μοτίβα συμβάλλοντας την αποτελεσματική αναγνώριση σχεδίων, εικόνων και αντικειμένων. Βέβαια για την ορθή αναγνώριση εικόνας απαιτούνται πολλαπλά επίπεδα και προσαρμογή των παραμέτρων τους. Παράμετροι όπως το βάθος του επιπέδου, η κίνηση των φίλτρων πάνω στις εικόνες και η μηδενική επικάλυψη ειναι κρίσιμοι για την αποτελεσματικότητα του κάθε επιπέδου και του ρόλου του στο γενικότερο δίκτυο.

Το συγκεντρωτικό επίπεδο, λειτουργεί ως επίπεδο μείωσης παραμέτρων ελαττώνοντας τις χωρικές διαστάσεις της εξόδου ενός συνελικτικού επιπέδου. Η εισαγωγή του συγκεκριμένου επιπέδου βοηθά στην βελτιστοποίηση του μοντέλου και διατήρηση δεδομένων. Επιπλέον, για την επιλογή συνάρτησης συγκέντρωσης, υπάρχουν 3 συνήθεις επιλογές.

- Max pooling απο όπου διατηρούνται οι μέγιστες τιμές ανα περιοχή διατηρώντας τα πιο σημαντικά χαρακτηριστικά.

- Average pooling με την οποία επιλέγεται ο μέσος όρος κάθε περιοχής διατηρώντας πολλαπλά χαρα.

- L2-norm με την οποία υπολογίζεται η ευκλείδεια νόρμα ανα περιοχή διατηρώντας μεγάλο μέρος των χαρακτηριστικών.

Τέλος το πλήρως συνδεδεμένο επίπεδο συναντάται στην έξοδο των συνελικτικών δικτύων ή μετά των συγκεντρωτικών.

Στα πλαισια της διπλωματικης αυτής, χρησιμοποιούμε τις διασημες αρχιτεκτονικές συνελικτικών δικτύων ResNet και MobileNetV2.

Η αρχιτεκτονική ResNet[34] αποτελείται απο ενα νεο σύνολο επιπέδων με την δυνατότητα παράκαμψης συνδέσεων. Με την τεχνική αυτή, τα δίκτυα διορθώνουν το πρόβλημα εξαφανιζόμενης κλίσης που προκύπτει απο την ύπαρξη πολλαπλών επιπέδων. Παράλληλα, επιτυγχάνουν την μείωση χωρικών διαστάσεων και την βελτίωση αποδόσεων σε σύγκριση με κανονικά συνελικτικά δίκτυα δημιουργώντας έτσι νεα ελαφρα αποδοτικά μοντέλα.

Η αρχιτεκτονική MobileNetV2[44] είναι ειδικά σχεδιασμένη για κινητές και ενσωματωμένες συσκευές περιορισμένων υπολογιστικών πόρων. Βασισμένη στην αρχιτεκτονική MobileNet[13], χρησιμοποιεί inverted residual block με γραμμικά επίπεδα συμφόρησης μειώνοντας το πλήθος των παραμέτρων και την υπολογιστική πολυπλοκότητά τους διατηρώντας την ισχύς τους. Χτίζοντας στην αρχιτεκτονική αυτή, το MobileNetV2 περιλαμβάνει επιπλέον επίπεδα γραμμικής συμφόρησης αποτελούμενα απο 1×1 συνελίξεις με γραμμική συνάρτηση ενεργοποίησης με σκοπό την ελάττωση απώλειας πληροφοριών και ενίσχυση ροής της κλίσης κατα την εκπαίδευση.

Επιτηδευμένες επιθέσεις

Οι επιτηδευμένες επιθέσεις εκμεταλλεύονται τις ευαισθησίες των Νευρωνικών Δικτύων παράγοντας παραπλανητικά παραδείγματα με ανεπαίσθητες διαταραχές προκαλώντας την λανθασμένη πρόβλεψη των δικτύων [52],[26].

Αναφερόμενες κυρίως σε εικόνες, οι επιθέσεις αυτές αφορούν και άλλους τύπους δεδομένων όπως κείμενα και γράφους.

Στην περίπτωση εικόνων, ο σκοπός των επιθέσεων ειναι η καθοδήγηση των δικτύων σε λανθασμένες προβλέψεις με υψηλά ποσοστά άνεσης κατατάσσοντας τες σε στοχευμένες και μη στοχευμένες.
Οι στοχευμένες επιθέσεις [40], απαιτούν τόσο την εισαγόμενη εικόνα όσο και το στοχευμένο αποτέλεσμα. Ο σκοπός τους ειναι η δημιουργία παραπλανητικών παραδειγμάτων που θα οδηγήσουν το δίκτυο στην λανθασμένη στοχευμένη πρόβλεψη.
Οι μη-στοχευμένες επιθέσεις απο την άλλη, απαιτούν μόνο την εισαγόμενη εικόνα. Σε αντίθεση με τις στοχευμένες, ο σκοπός τους ειναι η δημιουργία παραδειγμάτων που οδηγουν το δίκτυο σε μια γενική λανθασμένη πρόβλεψη.

Ανεξάρτητα απο τον τύπο επίθεσης, χωρίζονται επίσης σε ανοιχτού και κλειστού τύπου επιθέσεις.

Οι επιθέσεις ανοιχτού τύπου, απαιτουν πληρη γνώση του δικτύου, των παραμέτρων, της αρχιτεκτονικής και της κλίσης του. Εξετάζοντας τα δεδομένα αυτά εχουν την δυνατότητα να εκμεταλλευτούν τα πιο τρωτά σημεία του δικτύου για να παράξουν τα αντίστοιχα παραδείγματα γρήγορα και αποτελεσματικά.

Οι κλειστού τύπου επιθέσεις[48], δεν απαιτουν καμια πληροφορία για το δίκτυο πέρα απο τις τελικές προβλέψεις του. Με πολλαπλές συνεχής δοκιμές, επιχειρούν την εκτίμηση του πεδίου απόφασης του δικτύου παράγοντας έτσι τα αντίστοιχα παραδείγματα. Ενώ αυτή η διαδικασία ειναι δραματικά πιο χρονοβόρα απο τις επιθέσεις ανοιχτού τύπου, προσφέρει μια πιο ρεαλιστική όψη ως προς την ανθεκτικότητα του δικτύου και τείνει να ειναι πιο αποτελεσματική.

Εντός αυτής της κατηγορίας βρίσκεται και η **HopSkipJumpAttack**[17] την οποία χρησιμοποιούμε για την δημιουργία των παραδειγμάτων στην μεθοδολογία μας.

Προστασία απο επιθέσεις

Αρχικά ορίζουμε την ανθεκτικότητα ενος δικτύου ως την δυνατότητα εως ενα σημείο να μην παραπλανάται απο παραγόμενα παραδείγματα, προβλέποντας ορθά τις κλάσεις τους.

Για την ενίσχυση της ανθεκτικότητας των δικτύων έχουν προταθεί πολλαπλές μέθοδοι απο τις οποίες ακολουθούν ορισμένες πιο σημαντικές.

1. **Adversarial Training**

   Με την μέθοδο αυτή γίνεται η χρήση ενος αλγόριθμου επίθεσης για την δημιουργία παραδειγμάτων. Τα παραδείγματα αυτά χρησιμοποιούνται ως σύνολο δεδομένων στο οποιο επανεκπαιδεύεται το δίκτυο. Με τον τρόπο αυτό, το δίκτυο γενικεύει τα όρια απόφασής του για την κάθε κλάση ενισχύοντας την ανθεκτικότητά του. Είναι σημαντικό ωστόσο να σημειωθεί οτι αυτή η μέθοδος είναι τοσο αποτελεσματική όσο ο αλγόριθμος επίθεσης που χρησιμοποιείται και το μέγεθος των δεδομένων.

2. **Προ-επεξεργασία**

   Η προ-επεξεργασία των δεδομένων είναι ενας ακόμα αποδεδειγμενος τρόπος ενίσχυσης της ανθεκτικότητας των δικτύων. Καθώς τα παραγόμενα παραδείγματα είναι εικόνες με εισαγόμενο ανεπαίσθητο θόρυβο, η επεξεργασία τους με ποικίλους αλγορίθμους, δύναται να ελαττώσει ή να αφαιρέσει εξόλοκλήρου αυτές τις διαταραχές. Επομένως, η χρήση της προ-επεξεργασίας ειναι ενας αποτελεσματικός και γρήγορος τρόπος ενίσχυσης της ανθεκτικότητας των δικτύων. Ωστόσο, η χρήση αλγορίθμων σε περιπτώσεις συσκευών περιορισμένων υπολογιστικών πόρων προσθέτει υπολογιστική πολυπλοκότητα που πιθανώς δεν μπορει να υποστηριχθεί. Επιπλέον, αν και αποτελεσματική μέθοδος, δεν επηρεάζει το δίκτυο παρά περιορίζει την είσοδο του.

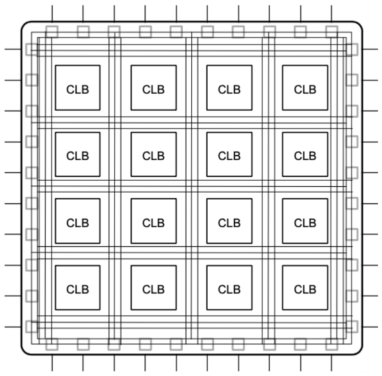
3. **Κβάντωση**

   Η Κβάντωση ειναι μια ακόμη μέθοδος προστασίας απο επιτηδευμένες επιθέσεις. Όπως αναλύσαμε παραπάνω, η Κβάντωση περιορίζει την ακρίβεια των παραμέτρων ενός μοντέλου για χρήση σε ειδικού σκοπού συσκευές. Λόγω του περιορισμού αυτού, το εύρος της απόφασης για την κάθε κλάση διευρύνεται ενισχύοντας έτσι την ανθεκτικότητά τους. Επιπλέον αξίζει να σημειωθεί πως η Κβάντωση επηρεάζει και την ανθεκτικότητα του δικτύου, ωστόσο, στα πλαίσια αυτής της εργασίας είναι απαιτούμενο στάδιο και η επίδρασή της συνυπολογίζεται στα αποτελέσματα.

FPGA

Τα Field-Programmable Gate Array (FPGAs) ειναι μια οικογένεια ημιαγώγιμων συσκευών αντίστοιχων των ολοκληρωμένων κυκλωμάτων που διαθέτουν προγραμματιζόμενα λογικά μπλοκ (CLB) και διαμορφώσιμες διασυνδέσεις. Αποτελούμενα απο τα μπλοκ αυτά, φέρουν επιπλέον μπλοκ ψηφιακής επεξεργασίας σήματος (DSP) και ενσωματωμένη μνήμη για την ενίσχυση των υπολογιστικών τους δυνατοτήτων 4.17.

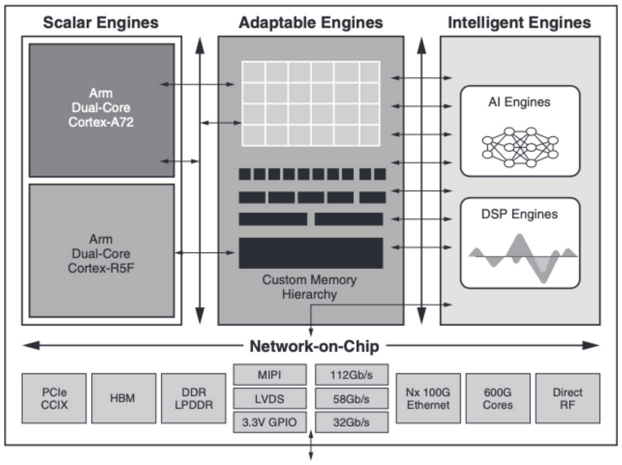
Τα FPGA μέσω της αναδιαμόρφωσης και του παραλληλισμού προσφέρουν ξεχωριστά πλεονεκτήματα στον τομέα της ΤΝ υποστηρίζοντας προσαρμόσιμες αρχιτεκτονικές. Με τον τρόπο αυτό ενισχύουν την αποδοτικότητα των νευρωνικών δικτύων παράλληλα με την βελτιστοποίηση πόρων. Επιπλέον, τον χαμηλό τους κόστος σε συνδυασμό με χαμηλή καθυστέρηση και υψηλές αποδόσεις τα καθιστά κατάλληλα για εφαρμογές σε Edge περιβάλλοντα.



Σχήμα 1.8: Παράδειγμα FPGA Αρχιτεκτονικής, Πηγή: [57]

Versal Adaptive Compute Acceleration Platform

SoC FPGAs ειναι μοντέρνου τύπου συσκευές αξιοποιώντας τις λειτουργίες των FPGA και των παραδοσιακών SoC προσφέροντας μια ευέλικτη πλατφόρμα ετερογενή υπολογισμών ικανή για πολλαπλές εφαρμογές. Μια τέτοια οικογένεια συσκευών ειναι η Versal Adaptive Compute Acceleration Platform (Versal ACAP)™ προσαρμοσμένες στην χρήση νευρωνικών δικτύων. Η οικογένεια αυτών των συσκευών ενσωματώνει προσαρμόσιμες μονάδες επεξεργασίας και μηχανες επιτάχυνσης με προγραμματιζόμενη λογική και συνδεσιμότητα προσφέροντας ετερογενής λύσεις υλικών για ενα ευρύ φάσμα εφαρμογών. Αυτές οι συσκευές συνδυάζουν μηχανές TN και DSP μαζί με προσαρμόσιμες και κλιμακωτές μηχανές σε ένα ολοκληρωμένο πακέτο Network-on-Chip (NoC) προσφέροντας εξαιρετικές αναλογίες απόδοσης-ισχύς συγκριτικά με τυπικούς επεξεργαστές και καρτες γραφικών.



Σχήμα 1.9: Διάγραμμα Xilinx Versal, Πηγή: [9]

Βασίζοντας πάνω στα πλεονεκτήματα των FPGA, τα Versal ACAP με την ετερογενή αρχιτεκτονική τους, ενισχύουν περαιτέρω την αποδοτικότητα και απόδοση των νευρωνικών δικτύων καθιστώντας τα εξαιρετικές επιλογές για αυτές τις εφαρμογές.

1.3 Προτεινόμενη Μεθοδολογία

Προτείνουμε την εφαρμογή σειράς απο εκπαιδεύσεις σε παραγόμενα παραδείγματα σε συνδυασμό με Κβάντωση για την ενίσχυση της ανθεκτικότητας των δικτύων ελέγχοντας τις επιδράσεις τους σε κάθε βήμα. Στην συνέχεια, μεταφέρουμε τα ενισχυμένα μοντελα στην πλατφόρμα Versal VCK190, ελέγχοντας για αλλαγές ως προς την απόδοση και την ανθεκτικότητα τους.

Σηματοθορυβικός Λόγος

Για την ποσοτικοποίηση της ανθεκτικότητας χρησιμοποιούμε την συνάρτηση Μέγιστου Σηματοθορυβικού λόγου (PSNR) που εκφράζεται απο την εξίσωση (1.12).

$$PSNR = 10 \cdot \log_{10}(\frac{MAX_I^2}{\sqrt{MSE}}) = 20 \cdot \log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{1.12}$$

Η εξίσωση αυτή αποτελείται απο τον λογαριθμικό λόγο της μέγιστης τιμής της εικόνας προς το τετράγωνο της διαφοράς μεταξύ της αρχικής και τελικής εικόνας εξ.(1.13).

$$\begin{gathered} MSE_c = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I_c(i,j) - K_c(i,j)]^2 \\ MSE_{total} = MSE_R + MSE_G + MSE_B \end{gathered} \tag{1.13}$$

Συνήθως χρησιμοποιείται για την εκτίμηση θορύβου σε αλγορίθμους συμπίεσης ωστόσο στην περίπτωση μας, έχουμε την δυνατότητα να μετρήσουμε τις διαφορές μεταξύ των αρχικών εικόνων και των αντίστοιχων παραδειγμάτων απο τις επιθέσεις. Στο σχήμα 1.10 παρατηρούμε ενα παράδειγμα αυτής της διαδικασίας και ποσοτικοποίησης των εικόνων.

(α′) Αρχική Εικόνα (β′) PSNR = 45.53dB (γ′) PSNR = 36.81dB (δ′) PSNR = 31.45dB

Σχήμα 1.10: Παράδειγμα χρήσης PSNR, Πηγή:[3]

Εργαλεία

Όπως εχουμε ήδη αναφέρει χρησιμοποιούμε της αρχιτεκτονικές ResNet20, ResNet56 και MobileNetV2 για τα μοντέλα μας λόγω της εργονομικής και ακριβείς φύσης τους. Τα μοντέλα αυτά επιλέχθηκαν επίσης λόγω της υποστήριξης και απο την πλατφόρμα Versal ελαχιστοποιώντας τυχών προβλήματα μετατροπής και εφαρμογής τους. Επιπροσθέτως, ενώ τα ResNet20 και ResNet56 ανήκουν στην ίδια οικογένεια μοντέλων, διαφοροποιούνται στο πλήθος των επιπέδων τους με 20 επίπεδα στο ResNet20 και 56 στο ResNet56. Βέβαια η ενισχυμένη πολυπλοκότητα τόσο του ResNet56 όσο και του MobileNetV2 οδηγούν σε αυξημένη απόδοση τόσο σε μεγάλα όσο και σε μικρά σύνολο δεδομένων.

Για σύνολα δεδομένων, αξιοποιούμε τα σύνολα CIFAR-10[38] για το οποίο έχουμε προ-εκπαιδευμένες εκδόσεις των μοντέλων μας και FashionMNIST[18] στο οποίο τα εκπαιδεύουμε απο την αρχή. Το CIFAR-10 αποτελείται απο 60000 έγχρωμες εικόνες 32×32 ανάλυσης ισομερώς κατανεμημένες σε 10 κλάσεις. Αποτελεί απο τα πιο διάσημα σύνολα δεδομένων στον κόσμο της μηχανικής μάθησης καθώς παρουσιάζει ποικιλομορφία δεδομένων διαφορετικής πολυπλοκότητας. Αξίζει επίσης να σημειωθεί πως τα δεδομένα του CIFAR-10 απαιτούν κανονικοποίηση την σημασία της οποίας θα εξηγήσουμε στα αποτελέσματα.
Το FashionMNIST αποτελείται απο 70000 ασπρόμαυρες εικόνες 28×28 ανάλυσης ισομερώς κατανεμημένες σε 10 κλάσεις. Εξελίσσοντας το σύνολο δεδομένων MNIST το οποίο αποτελούταν απο εικόνες χειρόγραφων αριθμών, το FashionMNIST φέρει πολλαπλούς τύπους ενδυμάτων προσφέροντας ενα πιο δύσκολο και σύνθετο σύνολο δεδομένων.

Για την εργασία μας στα νευρωνικά δίκτυα επιλέγουμε την χρήση της Python λόγω της απλότητας της και διευκόλυνσή μας μέσω υποστήριξης πολλαπλών πακέτων. Επιπλέον, επιλέγουμε το Pytorch Framework[4] για την ανάπτυξη των μοντέλων σε συνδυασμό με την βιβλιοθήκη Adversarial Robustness Toolbox[5] λόγω της υλοποίησης του HopSkipJumpAttack αλγορίθμου για την παραγωγή των παραπλανητικών παραδειγμάτων. Εν συνεχεία, για την κάλυψη των υπολογιστικών αναγκών αυτής της εργασίας, αξιοποιούμε τις υπηρεσίες Kaggle[2] και Google Colaboratory[1] που παρέχουν επεξεργαστές και κάρτες γραφικών μέσω Jupyter notebook.

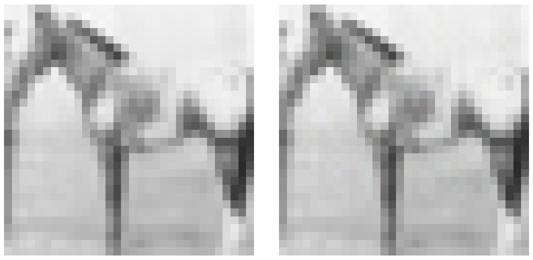
Υλοποίηση

Αρχικά όπως προαναφέραμε, εκπαιδεύουμε τα τρια μοντέλα στο FashionMNIST με χρήση του SGD βελτιστοποιητή επιτυγχάνοντας αποδόσεις 92.24%, 93.72%, 93.72% για τα ResNet20, ResNet56 και MobileNetV2 αντίστοιχα.

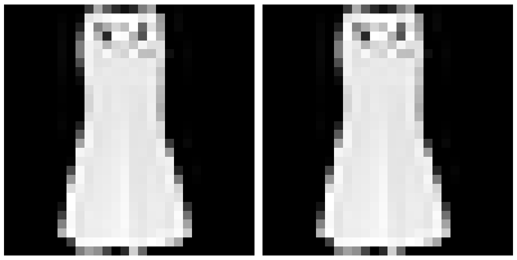
Ξεκινώντας την παραγωγή των παραπλανητικών παραδειγμάτων με την χρήση του HopSkipJumpAttack ορίζουμε τις παραμέτρους με τις προκαθορισμένες τιμές τους αποφεύγοντας την αλλοίωση της αποδοτικότητάς τους.

Η σειρά επανεκπαιδεύσεων αποτελείται απο τρεις διαδικασίες εκπαίδευσης παράγοντας νεα παραδείγματα σε κάθε βήμα. Πριν την εκκίνηση ωστόσο της σειράς επανεκπαιδεύσεων, παράγουμε μερικά παραδείγματα και με την χρήση του PSNR και του γεωμετρικού μέσου αυτών (εξ.(1.14)), έχουμε βασικές μετρήσεις με τις οποίες συγκρίνουμε την απόδοση των μεθόδων μας.

$$G_{mean} = \sqrt[n]{\prod_{i=1}^{n} x_i} \tag{1.14}$$


Αρχική - Άλογο 52.05 dB - Γάτα

(α΄) CIFAR-10 Παράδειγμα


Αρχική - Φόρεμα 51.71 dB - Παλτό

(β΄) FashionMNIST Παράδειγμα

Σχήμα 1.11: Παραδείγματα στο ResNet20

Επιπλέον, μέσω πολλαπλών ελέγχων και υπολογισμών βρήκαμε πως η διαφορά στον γεωμετρικό μέσο των PSNR μεταξύ 30000 και 400 εικόνων, είναι ±0.005dB. Η διαφορά αυτή παραβλέπεται και για μελλοντικές μετρήσεις παράγουμε μόνο 400 νεα παραδείγματα εξοικονομώντας χρόνο και πόρους.

Για την πρώτη επανεκπαίδευση επιλέγουμε πλήθος παραδειγμάτων 30.000 αποτελώντας 50% του CIFAR-10 και 42% του FashionMNIST. Επιλέγουμε ενα τόσο μεγάλο πλήθος ώστε να ωθήσουμε το μοντέλο σε μικρή μετακίνηση των ορίων απόφασης του, διατηρώντας την ακρίβειά του. Για την δεύτερη επανεκπαίδευση ωστόσο, ελαττώνουμε το πλήθος σε 20.000 αποτελώντας το 33% του CIFAR-10 και 28.5% του FashionMNIST. Η αλλαγή του πλήθους γίνεται τόσο για την εξοικονόμηση πόρων όσο και για την ενίσχυση της γενικοποίησης των μοντέλων. Τέλος, στην τρίτη επανεκπαίδευση χρησιμοποιούμε πλήθος 10.000 παραδειγμάτων αποτελώντας 16% του CIFAR-10 και 14% του FashionMNIST.

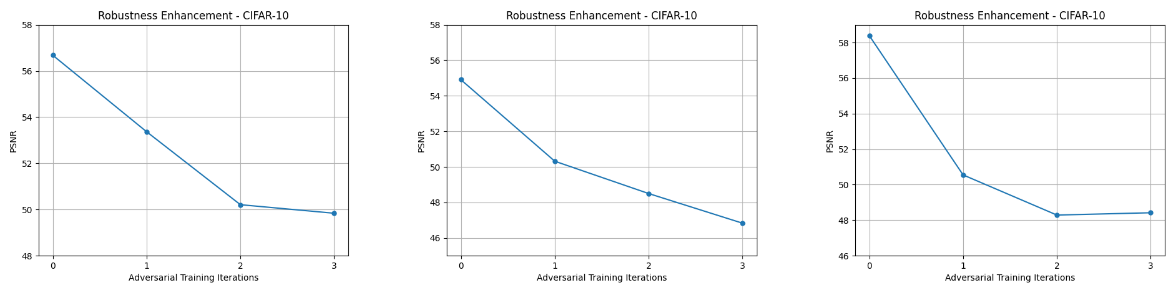
Κατα την κάθε επανεκπαίδευση επιτυγχάνουμε 99+% ακρίβεια στα παραγόμενα σύνολα δεδομένων με ελάχιστες πτώσεις στα αρχικά σύνολα. Επίσης αξίζει να σημειωθεί πως κατα την δεύτερη επανεκπαίδευση, αλλάξαμε βελτιστοποιητή απο SGD[7] σε Adam[36] καθώς η εκπαίδευση αποτύγχανε ελαχιστοποιώντας τόσο την ακρίβεια του νέου συνόλου όσο και του αρχικού. Τα αποτελέσματα αυτης της σειράς επανεκπαιδεύσεων εντοπίζονται στους πίνακες 1.1 και 1.2 για το CIFAR και FashionMNIST αντίστοιχα. Αντίστοιχα στα σχήματα 1.12 και 1.13 εντοπίζονται οι πτώσεις στο PSNR και στην ακρίβεια σε σχέση με την κάθε επανάληψη. Υπενθυμίζουμε οτι η πτώση στο PSNR υποδεικνύει την παρουσία ενισχυμένου θορύβου για την λανθασμένη πρόβλεψη των μοντέλων γεγονός που αποδεικνύει την ενισχυμένη ανθεκτικότητα.

	CIFAR-10			
	Αρχική Μέτρηση	Πρώτη εκπαίδευση	Δεύτερη Εκπαίδευση	Τρίτη Εκπαίδευση
ResNet20	56.68 dB	53.37 dB	50.21 dB	49.84 dB**
ResNet56	54.91 dB	50.37 dB	48.50 dB	46.83 dB
MobileNetV2	58.39 dB	50.55 dB	48.29 dB	48.42 dB

Πίνακας 1.1: Πτώση του PSNR ανά φορά επανεκπαίδευσης, CIFAR-10

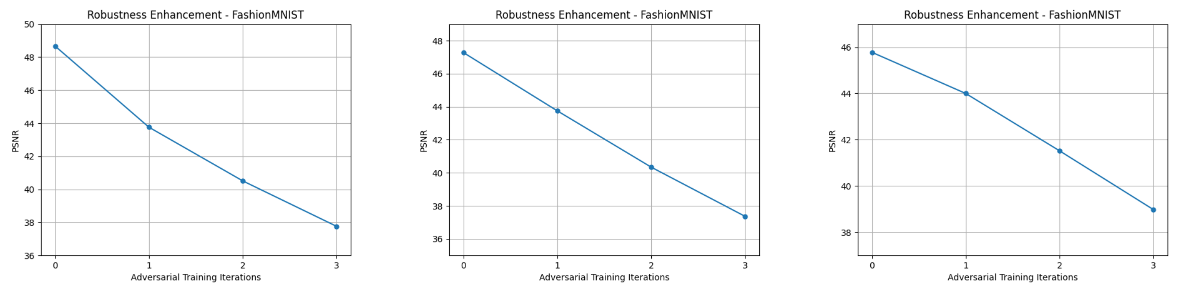
	FashionMNIST			
	Αρχική Μέτρηση	Πρώτη εκπαίδευση	Δεύτερη Εκπαίδευση	Τρίτη Εκπαίδευση
ResNet20	48.67 dB	43.76 dB	40.51 dB	37.76 dB
ResNet56	47.28 dB	43.75 dB	40.34 dB	37.36 dB
MobileNetV2	45.78 dB	44.00 dB	41.52 dB	38.98 dB

Πίνακας 1.2: Πτώση του PSNR ανά φορά επανεκπαίδευσης, FashionMNIST



(α′) ResNet20 Μοντέλο (β′) ResNet56 Μοντέλο (γ′) MobileNetV2 Μοντέλο

Σχήμα 1.12: Σταδιακή ενίσχυση ανθεκτικότητας - CIFAR-10



(α′) ResNet20 Μοντέλο (β′) ResNet56 Μοντέλο (γ′) MobileNetV2 Μοντέλο

Σχήμα 1.13: Σταδιακή ενίσχυση ανθεκτικότητας - FashionMNIST

Κβάντωση

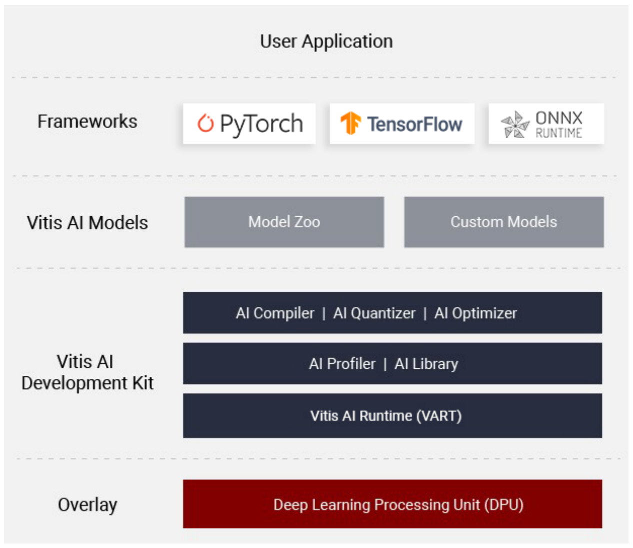
Για την εκτέλεση της Κβάντωσης εφαρμόζουμε Στατική Κβάντωση μετά την εκπαίδευση σε κάθε βήμα του μοντέλου, εξετάζοντας τις επιδράσεις της τόσο στην ακρίβεια όσο και στην ανθεκτικότητα. Ειναι σημαντικο επισης να σημειωθεί, πως για την εφαρμογή της Κβάντωσης απαιτήθηκαν αλλαγές στις αρχιτεκτονικές των μοντέλων και κυρίως η ενθυλάκωση τους σε περιβλήματα Κβάντωσης. Παράλληλα, ενω παρατηρήθηκε σημαντική αύξηση της ανθεκτικότητας των δικτύων, ως αποτέλεσμα της Κβάντωσης επιτεύχθηκε και συμπίεση των μοντέλων κατα 33% όπως φαίνεται στον πίνακα 1.3.

	Συμπίεση Μοντέλων	
	Αρχικό Μοντέλο	Κβαντισμένη έκδοση
ResNet20	1 Mbytes	355 kbytes
ResNet56	3.1 Mbytes	1.1 Mbytes
MobileNetV2	17.6 Mbytes	4.6 Mbytes

Πίνακας 1.3: Σύγκριση μεγεθών μεταξύ κανονικής και κβαντισμένης έκδοσης δικτύων

VitisAI - Versal

Τέλος, χρησιμοποιούμε την σουίτα εργαλείων VitisAI απο την Xilinx κατασκευασμένη για υλοποίηση και εφαρμογή νευρωνικών δικτύων σε FPGA και SoC της Xilinx. Η σουίτα αυτή παρέχει πολλαπλά εργαλεία μαζι και με βιβλιοθήκη αποτελούμενη απο πολλαπλά προ-εκπαιδευμένα μοντέλα έτοιμα και προσαρμοσμένα για την εφαρμογή τους στις συσκευές της εταιρίας. Επιπλέον, παρέχει και εργαλεία Κβάντωσης, βελτιστοποίησης και μεταγλώττισης μοντέλων. Στην περίπτωσή μας, εκμεταλλευόμαστε τα εργαλεία Κβάντωσης και μεταγλώττισης καθώς ειναι προαπαιτούμενα για την εφαρμογή των δικτύων μας στην πλατφόρμα Versal. Σε αντίθεση με την

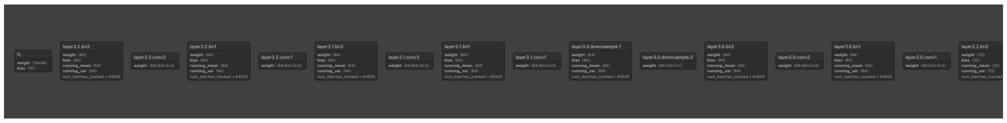


Σχήμα 1.14: Εργαλεία Vitis AI, Πηγή: [55]

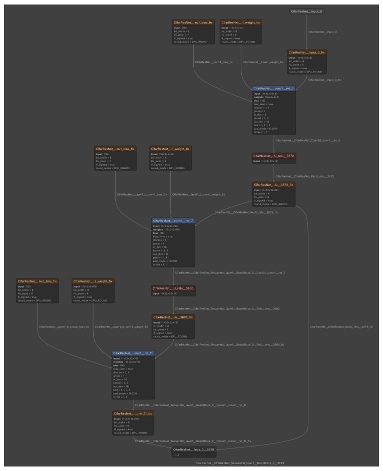
Κβάντωση που εφαρμόζουμε εμείς, το εργαλείο χρησιμοποιεί ιδιόκτητες βιβλιοθήκες και ορισμούς για το κάθε επίπεδο των μοντέλων στα οποία έπειτα εφαρμόζει την ίδια στατική Κβάντωση. Παράλληλα με αυτό το εργαλείο, χρησιμοποιούμε και τον παρεχόμενο μεταγλωττιστή για την μετατροπή των μοντέλων μας στην αρχιτεκτονική της συσκευής μας. Με την χρήση του προγράμματος Netron, παρατηρούμε την πορεία του μοντέλου, απο την αρχική του κατάσταση, στην κβαντισμένη απο το εργαλείο και τελική έτοιμη για χρήση στο σχήμα 1.15.

Εκτελώντας αυτή την διαδικασία για το κάθε μοντέλο, είμαστε έτοιμοι να μεταφερθούμε στην πλακέτα Versal VCK190 1.16 η οποία, με χρήση των οδηγιών του κατασκευή, έχει ρυθμιστεί κατάλληλα για την χρήση μας.

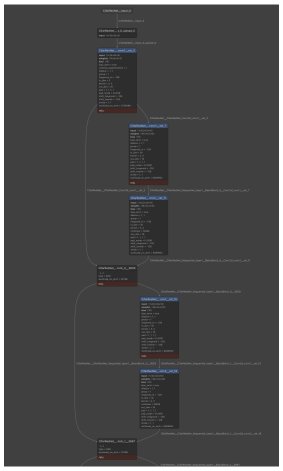
Σε αντίθεση με την εκτελούμενη μεχρι τώρα διαδικασία, η πλακέτα απαιτεί την χρήση C++ γλώσσας για την χρήση των μοντέλων και ως εκ τούτου δημιουργούμε τα αντίστοιχα αρχεία κώδικα για τον σκοπό αυτό. Επιπλέον, για την πρόβλεψη εικόνων και άντληση μετρήσεων δεν έχουμε την δυνατότητα χρήσης συνόλων δεδομένων επομένως εξάγουμε τις εικόνες απο τα σύνολα που ενδιαφερόμαστε και μέσω πολλαπλών ενεργειών αντλούμε τις απαραίτητες μετρήσεις απο την πλακέτα μας.



(α′) Αρχικό μοντέλο

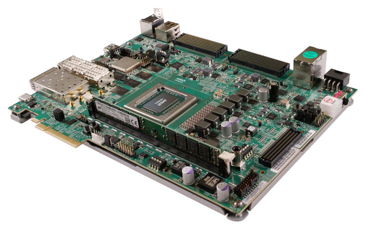


(β′) Κβαντισμένο Μοντέλο



(γ′) Τελικό μοντέλο

Σχήμα 1.15: Διαδικασία Κβάντωσης-Μετατροπής



Σχήμα 1.16: Versal VCK190 Board, Πηγή: [56]

1.4 Αποτελέσματα

Αναλύοντας τα αποτελέσματά μας, πρέπει να σημειωθεί πως για λόγους απλότητας οι μετρήσεις χωρίζονται ανά σύνολο δεδομένων. Όπως αναφέραμε παραπάνω ξεκινάμε με αρχικές μετρήσεις στους πίνακες ;; και ;;, τις οποίες θα χρησιμοποιήσουμε για την ποσοστιαία έκφραση των αποτελεσμάτων μας.

	CIFAR-10	
	Ακρίβεια	PSNR
ResNet20	92.60 %	56.68 dB
ResNet56	94.37%	54.91 dB
MobileNetV2	94.22%	58.36 dB

Πίνακας 1.4: Αρχικές Μετρήσεις, CIFAR-10

	FashionMNIST	
	Ακρίβεια	PSNR
ResNet20	92.24%	48.67 dB
ResNet56	93.72%	47.28 dB
MobileNetV2	93.72%	45.78 dB

Πίνακας 1.5: Αρχικές Μετρήσεις, FashionMNIST

Η επανεκπαίδευση όπως αναλύσαμε εκτελείται σε τρία στάδια 30.000, 20.000 και 10.000 παραδειγμάτων έπειτα της οποίας ακολουθεί η Κβάντωση των μοντέλων.

CIFAR-10

Εξετάζοντας τους πίνακες 1.6 έως 1.8 παρατηρούμε μέγιστη πτώση ακρίβειας κατα 1.25% για το ResNet20, 2% για το ResNet56 και 1.4% για το MobileNetV2. Όπως αντιλαμβανόμαστε, οι πτώσεις αυτές είναι αρκετά μικρές για να μπορούν να θεωρηθούν αμελητέες ενώ οι διαφορές μεταξύ των μοντέλων υποδεικνύουν τις αντίστοιχη πολυπλοκότητά τους.

Εξετάζοντας και τους πίνακες 1.9 έως 1.11 παρατηρούμε την σημαντική πτώση στο PSNR του κάθε μοντέλου με 12.1% για το ResNet20, 14.71% για το ResNet56 και 14.07% για το MobileNetV2.

	ResNet20			
	Ακρίβεια	Διαφορά		
Αρχικές Μετρήσεις	92.60 %			
Πρώτη Επανεκπαίδευση	92.54 %	-0.06 %		
Δεύτερη Επανεκπαίδευση	92.00 %	-0.60 %	-0.54 %	
Τρίτη Επανεκπαίδευση	91.35 %	-1.25 %	-1.19 %	-0.65 %

Πίνακας 1.6: ResNet20 Εξέλιξη ακρίβειας, CIFAR-10

	ResNet56			
	Ακρίβεια	Διαφορά		
Αρχική Μέτρηση	94.37 %			
Πρώτη Επανεκπαίδευση	93.90 %	-0.47 %		
Δεύτερη Επανεκπαίδευση	93.14 %	-1.23 %	-0.76 %	
Τρίτη Επανεκπαίδευση	92.33 %	-2.04 %	-1.57 %	-0.81 %

Πίνακας 1.7: ResNet56 Εξέλιξη ακρίβειας, CIFAR-10

	MobileNetV2			
	Ακρίβεια	Διαφορά		
Αρχική Μέτρηση	94.22 %			
Πρώτη Επανεκπαίδευση	94.17 %	-0.05 %		
Δεύτερη Επανεκπαίδευση	93.58 %	-0.64 %	-0.59 %	
Τρίτη Επανεκπαίδευση	92.79 %	-1.43 %	-1.38 %	-0.79 %

Πίνακας 1.8: MobileNetV2 Εξέλιξη ακρίβειας, CIFAR-10

	ResNet20			
	PSNR	Πτώση μετρήσεων		
Βασε	56.68 dB			
Πρώτη Επανεκπαίδευση	53.37 dB	-5.83 %		
Δεύτερη Επανεκπαίδευση	50.21 dB	-11.41 %	-5.92 %	
Τρίτη Επανεκπαίδευση	49.84 dB**	-12.10 %	-6.65 %	-0.77 %

Πίνακας 1.9: ResNet20 Μέτρηση PSNR, CIFAR-10

	ResNet56			
	PSNR	Πτώση μετρήσεων		
Αρχική Μέτρηση	54.91 dB			
Πρώτη Επανεκπαίδευση	50.37 dB	-8.26 %		
Δεύτερη Επανεκπαίδευση	48.50 dB	-11.67 %	-3.71 %	
Τρίτη Επανεκπαίδευση	46.83 dB	-14.71 %	-7.02 %	-3.44 %

Πίνακας 1.10: ResNet56 Μέτρηση PSNR, CIFAR-10

	MobileNetV2			
	PSNR	Πτώση μετρήσεων		
Αρχική Μέτρηση	54.45 dB			
Πρώτη Επανεκπαίδευση	50.55 dB	-7.16 %		
Δεύτερη Επανεκπαίδευση	48.29 dB	-11.3 %	-4.47 %	
Τρίτη Επανεκπαίδευση	46.79 dB	-14.07 %	-7.43 %	-3.1 %

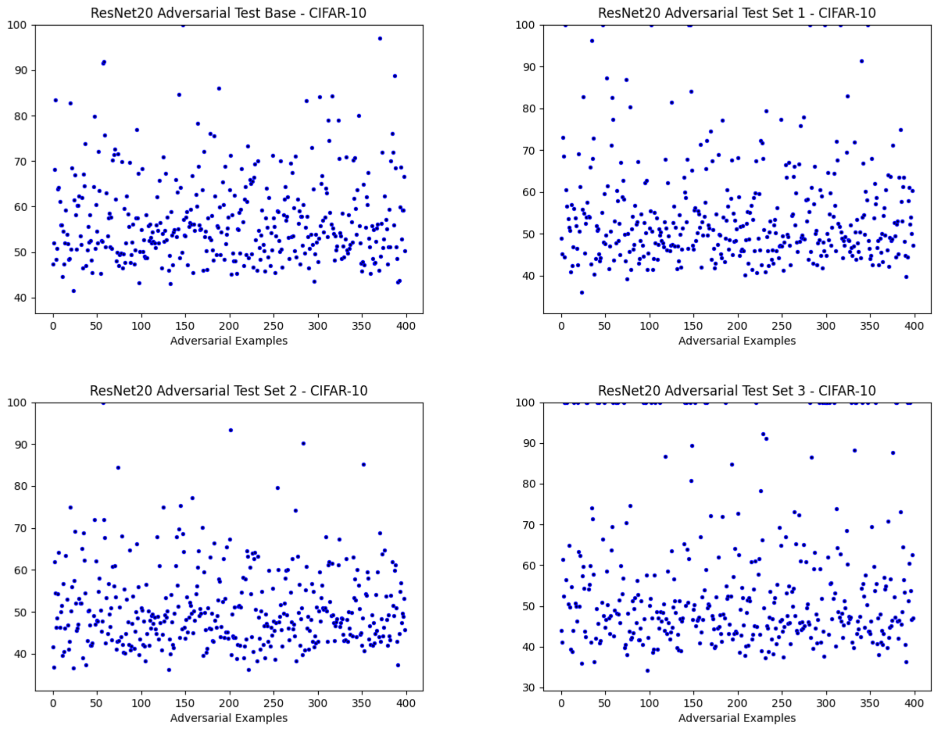
Πίνακας 1.11: MobileNetV2 Μέτρηση PSNR, CIFAR-10

Αν και μικρές βελτιώσεις σε σχέση με τους στόχους μας, ειναι αρκετά σημαντικές για να προχωρήσουμε την διαδικασία μας. Βέβαια για των υπολογισμό αυτών των μετρήσεων χρησιμοποιήσαμε τον γεωμετρικό μέσο όλων των PSNR του κάθε συνόλου. Το πλεονέκτημα αυτου υπολογισμού ειναι η δυνατότητα παράβλεψης απότομων αλλαγών στα δεδομένα ωστόσο ακριβώς για τον λόγο αυτό θα πρέπει να εξετάσουμε και της κατανομές τους. Στα σχήματα 1.17 έως 1.19, παρατηρούμε τις κατανομές των εικόνων σε σχέση με τα PSNR τους.

Όπως παρατηρούμε, με την πάροδο επαναλήψεων, τα δεδομένα/εικόνες προσαρμόζονται όλο και περισσότερο στις χαμηλότερες τιμές. Παράλληλα, εντοπίζουμε πιο μαζεμένες τιμές στα μοντέλα ResNet56 και MobileNetV2 τα οποια εκμεταλλευόμενα τις πιο σύνθετες αρχιτεκτονικές τους αποδεικνύονται πιο ανθεκτικά.

Αξιολογώντας και την Κβάντωση των μοντέλων, παρατηρούμε στον πίνακα 1.12 τις αναμενόμενες πτωσεις στην ακρίβεια.

Με σχετικά αυξημένες τελικές πτώσεις ακρίβειας λόγω της Κβάντωσης των μοντέλων, 1.32%, 2.13%, 1.85% για το κάθε μοντέλο αντίστοιχα, παρατηρούμε στον πίνακα 1.13 δραματικές πτώσεις στο PSNR γεγονός που όπως αναλύσαμε μεταφράζεται σε ενίσχυση ανθεκτικότητας. Όπως και πριν, αναλύοντας τις κατανομές των μοντέλων



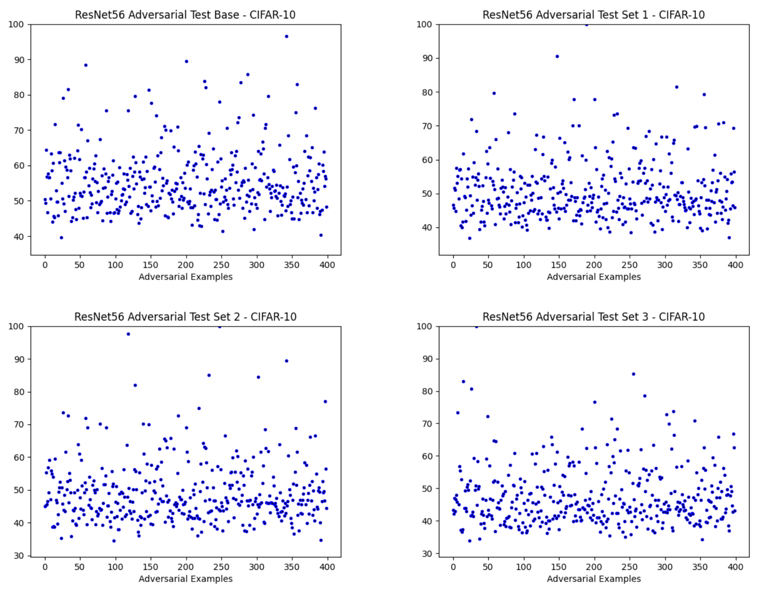
Σχήμα 1.17: ResNet20 κατανομή PSNR - CIFAR-10

	Ακρίβεια Κβαντισμένων μοντέλων			
	Αρχική Κβάντωση	Κβάντωση 1ου βήματος	Κβάντωση 2ου βήματος	Κβάντωση 3ου βήματος
ResNet20	92.33 % -0.27%	92.17 % -0.37%	91.87 % -0.13%	91.28 % -0.07%
ResNet56	94.22 % -0.15%	93.64 % -0.25%	92.81 % -0.33%	92.24 % -0.09%
MobileNetV2	94.03 % -0.19%	93.61 % -0.56%	93.28 % -0.30%	92.37 % -0.42%

Πίνακας 1.12: Ακρίβεια κβαντισμένων μοντέλων, CIFAR-10

	PSNR Κβανιτσμένων Μοντέλων			
	Αρχική Κβάντωση	Κβάντωση 1ου βήματος	Κβάντωση 2ου βήματος	Κβάντωση 3ου βήματος
ResNet20	39.41 dB -30.00%	35.46 dB -33.55%	32.85 dB -34.57%	32.64 dB -34.51%
ResNet56	38.18 dB -30.04%	34.46 dB -31.58%	32.58 dB -32.82%	31.31 dB -33.14%
MobileNetV2	41.45 dB -23.87%	33.39 dB -33.94%	31.43 dB -34.91%	32.18 dB -31.22%

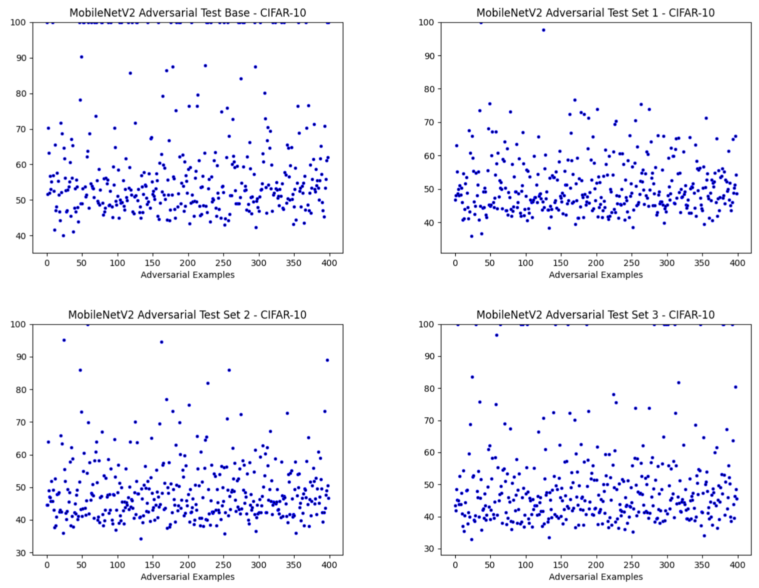
Πίνακας 1.13: Μέτρηση PSNR Κβαντισμένων μοντέλων, CIFAR-10



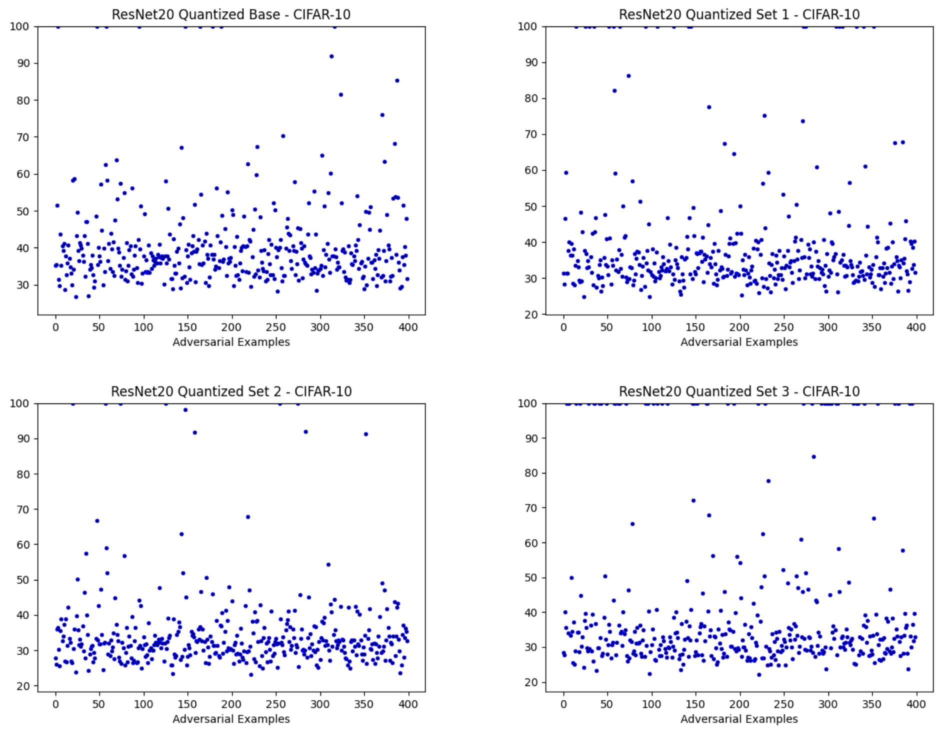
Σχήμα 1.18: ResNet56 κατανομή PSNR - CIFAR-10

στα σχήματα 1.20 έως 1.22, παρατηρούμε την δραματική πτώση τιμών αλλα και την σχεδον πλήρη συμμόρφωση των παραδειγμάτων στις χαμηλότερες τιμές. Ωστόσο, παρατηρούμε επίσης αρκετά παραδείγματα ακριβώς στα 100 dB. Αυτ·ο οφείλεται τόσο στην μειωμένη ακρίβεια του εκάστοτε μοντέλου όσο και στην αποτυχία του αλγορίθμου να παράξει παραδείγματα εντός των ορίων του.

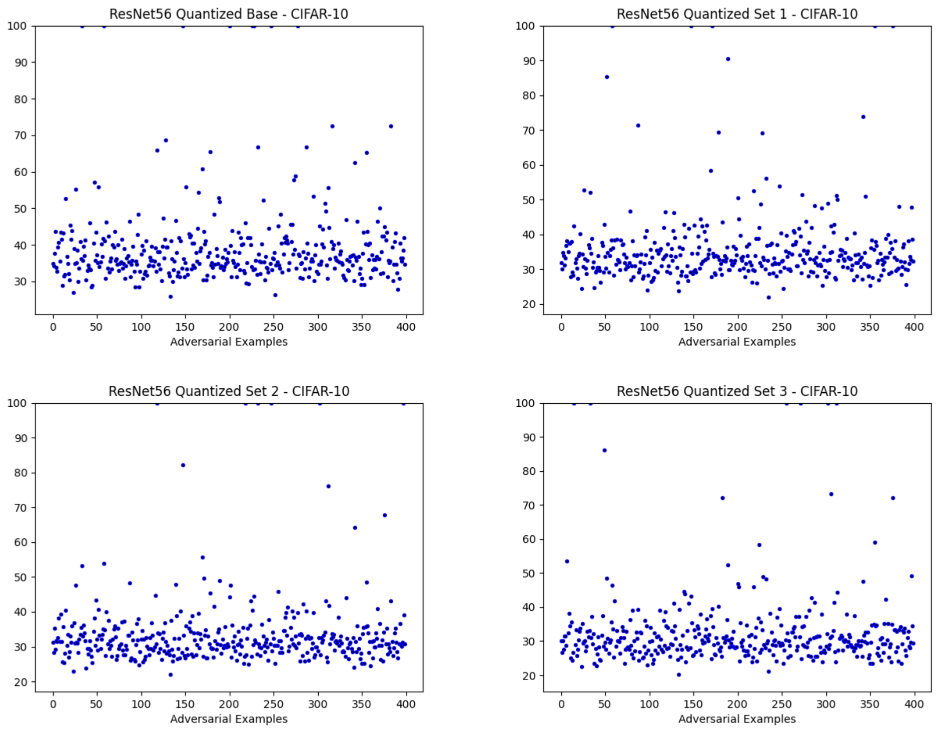
Τέλος στο σχήμα 1.23 μπορούμε να παρατηρήσουμε την εξέλιξη των μοντέλων μεσω αυτής της διαδικασίας ενώ το σχήμα 1.24 παρέχει το πλέον απαιτούμενο παράδειγμα για την παραπλάνηση του μοντέλου, στο οποίο ο προστιθέμενος θόρυβος είναι εύκολα αντιληπτός, επιβεβαιώνοντας την αποτελεσματικότητα της μεθόδου μας.



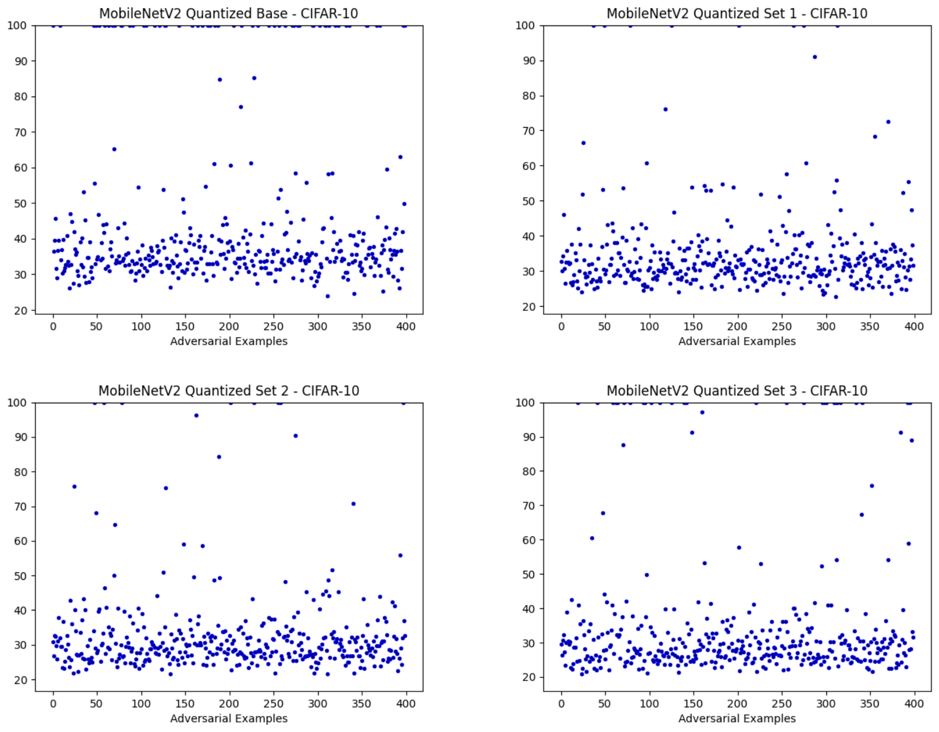
Σχήμα 1.19: MobileNetV2 κατανομή PSNR - CIFAR-10



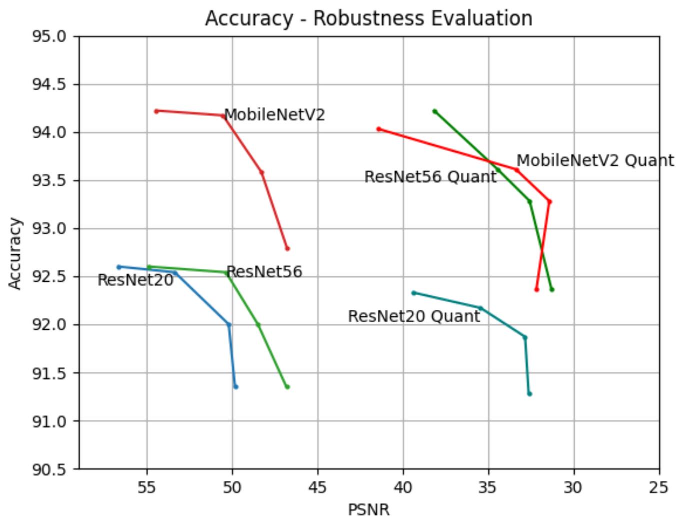
Σχήμα 1.20: Κβαντισμένο ResNet20 Κατανομή PSNR - CIFAR-10



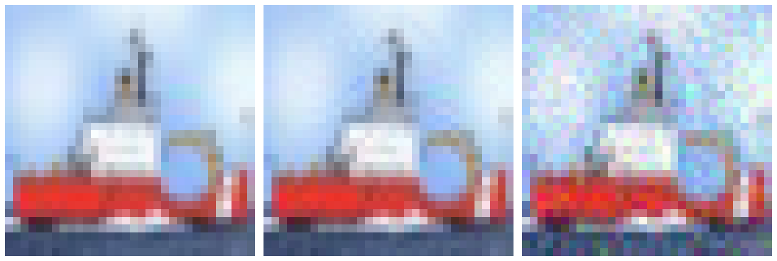
Σχήμα 1.21: Κβαντισμένο ResNet56 Κατανομή PSNR - CIFAR-10



Σχήμα 1.22: Κβαντισμένο MobileNetV2 Κατανομή PSNR - CIFAR-10



Σχήμα 1.23: Ακρίβεια - Ανθεκτικότητα - CIFAR-10



(α′) Αρχική εικόνα (β′) Δεύτερου συνόλου (γ′) Τελικό παράδειγμα

Σχήμα 1.24: ResNet20 Παράδειγμα ανθεκτικότητας - CIFAR-10

FashionMNIST

Εφαρμόζοντας τις ιδιες διαδικασίες για αυτό το σύνολο δεδομένων, αναμένουμε αυξημένες αποδόσεις σε μορφή υψηλής ακρίβειας και ελαττωμένου PSNR, συγκριτικά με το CIFAR-10 κυρίως λόγω τις απλούστερης φύσης του. Στους πίνακες 1.14 έως 1.16 παρατηρούμε τις πτώσεις ακρίβειας ενω στους πίνακες 1.17 έως 1.19 την βελτίωση της ανθεκτικότητας ανα μοντέλο.

	ResNet20			
	Ακρίβεια	Διαφορά		
Αρχική Μέτρηση	92.54 %			
Πρώτη Επανεκπαίδευση	93.52 %	+0.99 %		
Δεύτερη Επανεκπαίδευση	93.35 %	+0.81 %	-0.17 %	
Τρίτη Επανεκπαίδευση	93.01 %	+0.47 %	-0.51 %	-0.34 %

Πίνακας 1.14: ResNet20 Εξέλιξη Ακρίβειας, FashionMNIST

	ResNet56			
	Ακρίβεια	Διαφορά		
Αρχική Μέτρηση	93.72 %			
Πρώτη Επανεκπαίδευση	93.64 %	-0.08 %		
Δεύτερη Επανεκπαίδευση	93.63 %	-0.09 %	-0.01 %	
Τρίτη Επανεκπαίδευση	93.29 %	-0.43 %	-0.35 %	-0.34 %

Πίνακας 1.15: ResNet56 Εξέλιξη ακρίβειας, FashionMNIST

Αναλύοντας τα δεδομένα στην περίπτωση αυτή έχουμε μηδαμινές τελικές πτώσεις 0.43% και 0.41% για το ResNet56 και MobileNetV2 ενω το ResNet20 παρουσιάζει αύξηση της ακρίβειας. Αυτό βέβαια σχετίζεται με την αρχική εκπαίδευση του μοντέλου μας υποδεικνύοντας την δυνατότητά του για ενισχυμένη ακρίβεια. Παράλληλα, εξετάζοντας τις αλλαγές στο PSNR παρατηρούμε σημαντικές πτώσεις ύψους 22%, 21% και 15% για ResNet20, ResNet56 και MobileNetV2 αντίστοιχα όπως αναμέναμε. Επιπλέον αξίζει να δοθεί έμφαση στην διαφορά των αρχικών τιμών σε σύγκριση με το CIFAR-10. Καθώς το FashionMNIST ειναι μονοδιάστατο με συγκριτικά απλές εικόνες, τα μοντέλα έχουν την δυνατότητα καλύτερης προσαρμογής των πεδίων αποφάσεών τους καθιστώντας τα πιο ανθεκτικά.

	MobileNetV2			
	Ακρίβεια	Διαφορά		
Αρχική Μέτρηση	93.72 %			
Πρώτη Επανεκπαίδευση	93.52 %	-0.20 %		
Δεύτερη Επανεκπαίδευση	93.40 %	-0.32 %	-0.12 %	
Τρίτη Επανεκπαίδευση	93.31 %	-0.41 %	-0.21 %	-0.09 %

Πίνακας 1.16: MobileNetV2 Εξέλιξη Ακρίβειας, FashionMNIST

	ResNet20			
	PSNR	Πτώση Μετρήσεων		
Αρχική Μέτρηση	48.67 dB			
Πρώτη Επανεκπαίδευση	43.76 dB	-10.08 %		
Δεύτερη Επανεκπαίδευση	40.51 dB	-16.76 %	-7.42 %	
Τρίτη Επανεκπαίδευση	37.76 dB	-22.41 %	-13.71 %	-6.78 %

Πίνακας 1.17: ResNet20 Μέτρηση PSNR, FashionMNIST

	ResNet56			
	PSNR	Πτώση Μετρήσεων		
Αρχική Μέτρηση	47.28 dB			
Πρώτη Επανεκπαίδευση	43.75 dB	-7.46 %		
Δεύτερη Επανεκπαίδευση	40.34 dB	-14.67 %	-7.79 %	
Τρίτη Επανεκπαίδευση	37.36 dB	-20.98 %	-14.06 %	-7.38 %

Πίνακας 1.18: ResNet56 Μέτρηση PSNR, FashionMNIST

	MobileNetV2			
	PSNR	Πτώση Μετρήσεων		
Αρχική Μέτρηση	45.78 dB			
Πρώτη Επανεκπαίδευση	44.00 dB	-3.88 %		
Δεύτερη Επανεκπαίδευση	41.52 dB	-9.30 %	-5.63 %	
Τρίτη Επανεκπαίδευση	38.98 dB	-14.85 %	-11.40 %	-6.11 %

Πίνακας 1.19: MobileNetV2 Μέτρηση PSNR, FashionMNIST

Οι κατανομές αντίστοιχα των μοντέλων, παραλείπονται καθώς θα αναλυθούν και στις κβαντισμένες εκδόσεις, παρουσιάζουν ενισχυμένες σταδιακές προσαρμογές των παραδειγμάτων με επιπρόσθετα αποτυχημένα παραδείγματα στις τελικές καταστάσεις των μοντέλων.

Κβαντίζοντας και αυτές τις εκδόσεις των μοντέλων, παρατηρούμε στους πίνακες 1.20 και 1.21 τις αλλαγές ως προς την ακρίβεια και το μέσο PSNR τους.

	Ακρίβεια κβαντισμένων μοντέλων			
	Αρχική Κβάντωση	Κβάντωση 1ου βήματος	Κβάντωση 2ου βήματος	Κβάντωση 3ου βήματος
ResNet20	92.21 % -0.03%	93.44 % -0.08%	93.27 % -0.08%	92.97 % -0.04%
ResNet56	93.74 % +0.02%	93.60 % -0.04%	93.45 % -0.18%	93.12 % -0.17%
MobileNetV2	93.59 % -0.13%	93.28 % -0.24%	93.11 % -0.29%	93.15 % -0.16%

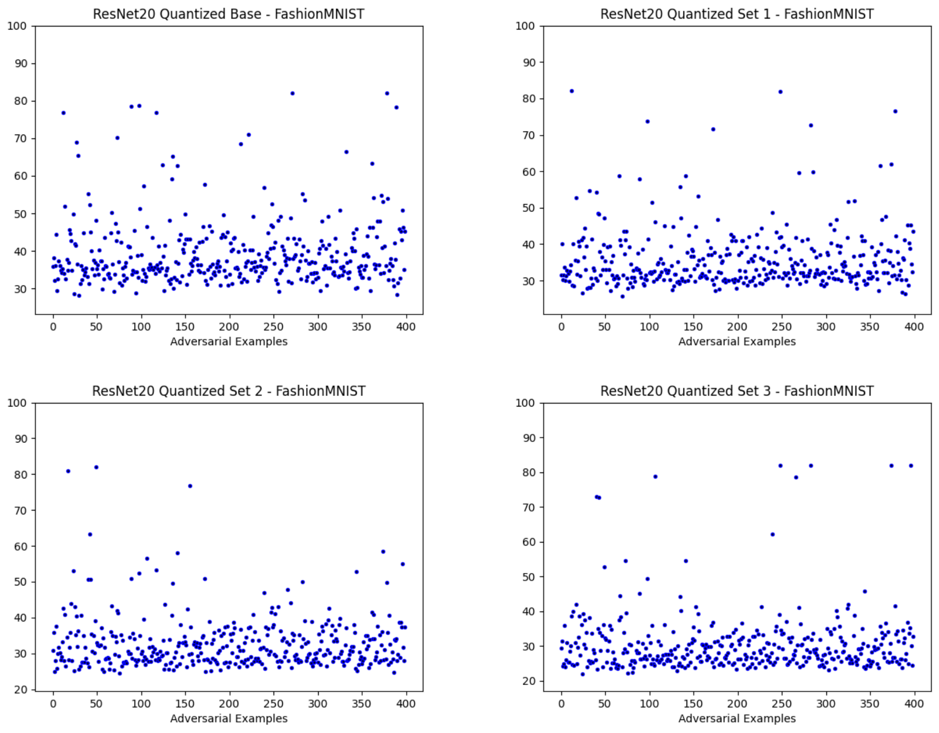
Πίνακας 1.20: Ακρίβεια κβαντισμένων μοντέλων, FashionMNIST

	PSNR Κβαντισμένων μοντέλων			
	Αρχική Κβάντωση	Κβάντωση 1ου βήματος	Κβάντωση 2ου βήματος	Κβάντωση 3ου βήματος
ResNet20	38.68 dB -20.50%	35.05 dB -19.90%	32.20 dB -20.51%	29.65 dB -21.47%
ResNet56	38.05 dB -19.52%	34.78 dB -20.50%	32.20 dB -20.17%	29.43 dB -21.22%
MobileNetV2	30.53 dB -33.33%	29.16 dB -33.72%	27.32 dB -34.20%	24.89 dB -36.14%

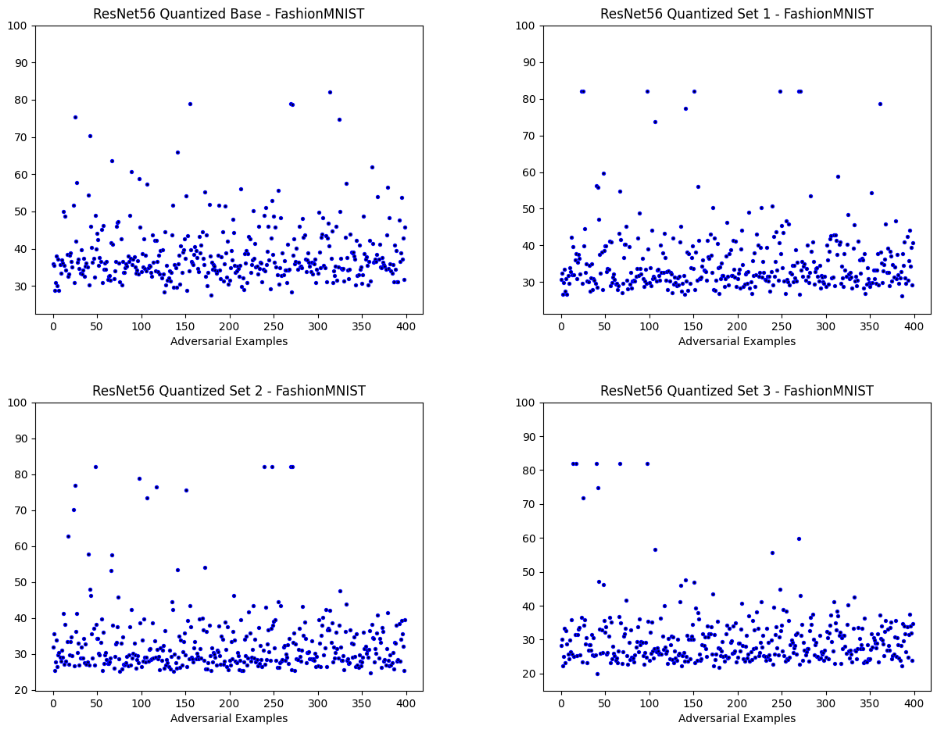
Πίνακας 1.21: Μέτρηση PSNR κβαντισμένων μοντέλων, FashionMNIST

Σε αντίθεση με το CIFAR-10 αλλα παρόμοια με τις μη-κβαντισμένες εκδόσεις, παρατηρούμε πτώσεις 0.6% και 0.57% για τα ResNet56 και MobileNetV2 ενώ, όπως αναλύσαμε και πριν, το ResNet20 παρουσιάζει ακόμα μεγαλύτερη αύξηση ακρίβειας. Επιπλέον παρατηρούμε πτώσεις του PSNR ύψους 33% και στα 3 μοντέλα με τελικές τιμες γύρω στα 24-29 dB. Εξάγοντας τις εικόνες του συνόλου, παρατηρούμε επίσης πως αυτο το εύρος στο PSNR οδηγεί σε πλήρη αναγνώριση ύπαρξης θορύβου απο ανθρώπους.

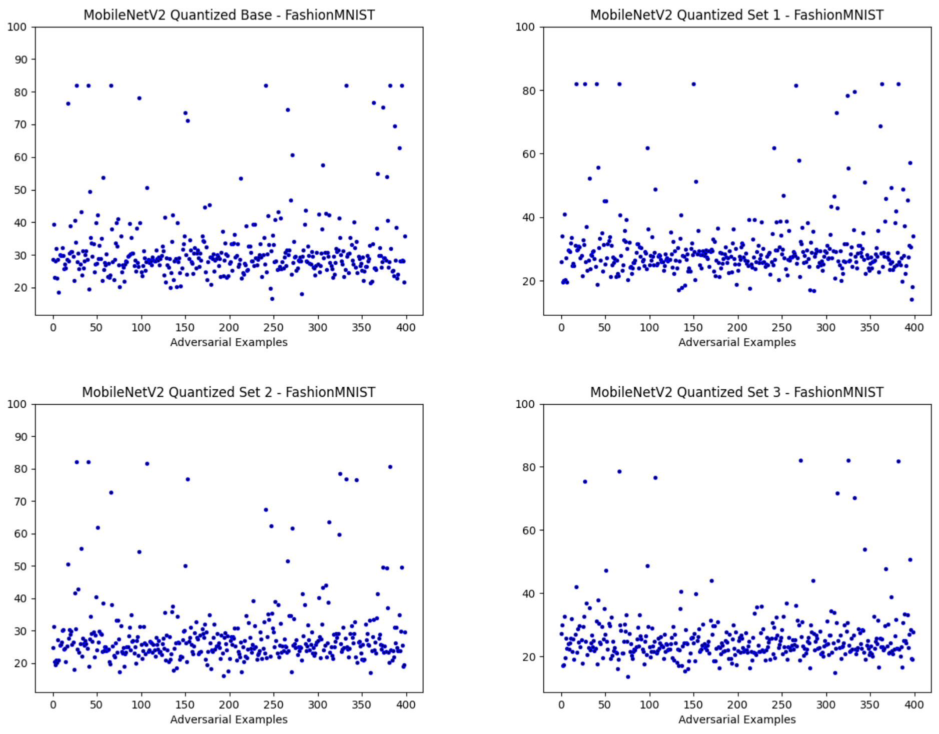
Στα σχήματα 1.25 έως 1.27 εντοπίζουμε τις κατανομές των PSNR τιμών για το κάθε μοντέλο όπου παρόμοια με τις προηγούμενα κβαντισμένα μοντέλα, παρατηρούνται σημαντικές συμμορφώσεις στις κατώτερες τιμές. Παράλληλα, παρατηρούμε πως μόνο 3% των παραδειγμάτων διατηρεί συγκριτικά υψηλές τιμές που μπορούν να θεωρηθούν ανεπαίσθητες, εκφράζοντας έτσι την δυνατότητα των επιλεγμένων δικτύων να διατηρήσουν την ανθεκτικότητα τους ακόμα και σε σύνθετα παραδείγματα.



Σχήμα 1.25: ResNet20 Κατανομή PSNR - FashionMNIST

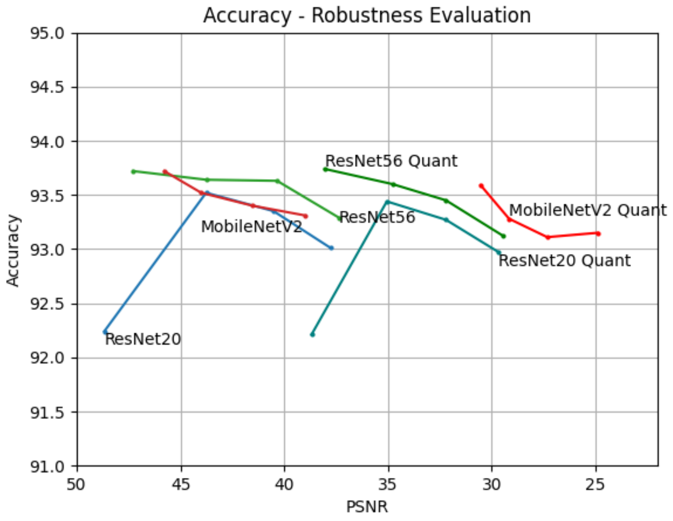


Σχήμα 1.26: ResNet56 Κατανομή PSNR - FashionMNIST

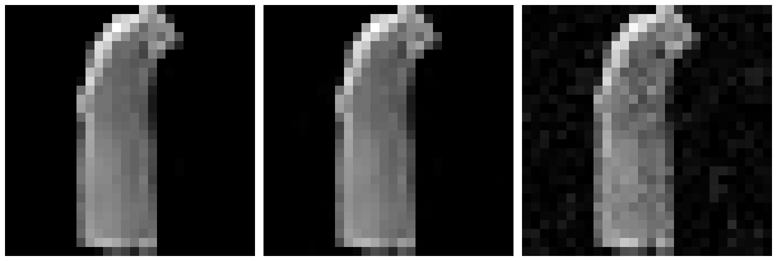


Σχήμα 1.27: MobileNetV2 Κατανομή PSNR - FashionMNIST

Τέλος στο σχήμα 1.28 περιλαμβάνεται το διάγραμμα εξέλιξης των μοντέλων ως προς την ακρίβεια και την ανθεκτικότητα σε κάθε στάδιο ενώ στο σχήμα 1.29 παρατηρούμε τον εμφανή απαιτούμενο θόρυβο των παραπλανητικών παραδειγμάτων.



Σχήμα 1.28: Ακρίβεια - Ανθεκτικότητα - FashionMNIST



(α΄) Αρχική Εικόνα (β΄) Δεύτερο σύνολο (γ΄) Τελική εικόνα

Σχήμα 1.29: ResNet20 Παράδειγμα ανθεκτικότητας - FashionMNIST

Versal

Έχοντας επιτυχώς εφαρμόσει την προτεινόμενη μεθοδολογία μας και έχοντας αντλήσει ικανοποιητικά αποτελέσματα, μεταφέρουμε τα μοντέλα στην πλακέτα μας εξετάζοντας τις επιδράσεις και αποδόσεις της. Για τον σκοπό αυτό χρησιμοποιούμε μόνο τις τελικές επεξεργασμένες εκδόσεις των μοντέλων κβαντισμένες απο το VitisAI.

Στους πίνακες 1.22 και 1.23 παρατηρούμε τις αποδόσεις για το CIFAR-10 ενω στους πίνακες 1.24 και 1.25 για το FashionMNIST.

		CIFAR-10 Ακρίβεια					
		Αρχικό Μοντέλο		Κβαντισμένο		**Versal**	
		Top - 1	Top - 5	Top - 1	Top - 5	Top - 1	Top - 5
ResNet20	Ακρίβεια	91.35 %	99.56 %	91.28 %	99.58 %	91.33 %	99.55 %
ResNet56	Ακρίβεια	92.33 %	99.67 %	92.24 %	99.62 %	92.13 %	99.57 %
MobileNetV2	Ακρίβεια	92.79 %	99.77 %	92.37 %	99.77 %	92.86 %	99.73 %

Πίνακας 1.22: Ακρίβεια μοντέλων στο Versal, CIFAR-10

		CIFAR-10 Απόδοση					
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
ResNet20	Καθυστέρηση	43.36 υς	25.31 υς	19.71 υς	18.55 υς	17.54 υς	20.89 υς
	Μέσα FPS	23 062	39 504	50 722	53 893	57 010	47 852
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
ResNet56	Καθυστέρηση	61.33 υς	35.47 υς	35.17 υς	35.31 υς	35.23 υς	35.28 υς
	Μέσα FPS	16 303	28 188	28 429	28 320	28 381	28 342
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
MobileNetV2	Καθυστέρηση	107.13 υς	81.13 υς	81.10 υς	81.05 υς	80.98 υς	81.20 υς
	Μέσα FPS	9 334	12 325	12 330	12 338	12 348	12 315

Πίνακας 1.23: Απόδοση μοντέλων στο Versal, CIFAR-10

Συγκρίνοντας αυτά τα αποτελέσματα, παρατηρούμε οτι η ακρίβεια των μοντέλων έχει αυξηθεί στα επίπεδα των αρχικών προ-κβαντισμένων σταδίων τους. Επιπλέον, παρατηρούμε την αυξημένη απόδοση εως 250% του ResNet20 με την χρήση πολλαπλών νημάτων σε αντίθεση με τα υπόλοιπα μοντέλα τα οποία δεν εμφανίζουν καμία βελτίωση με την χρήση περισσότερων των 2 νημάτων.

		FashionMNIST Ακρίβεια					
		Αρχικό Μοντέλο		Κβαντισμένο		**Versal**	
		Top - 1	Top - 5	Top - 1	Top - 5	Top - 1	Top - 5
ResNet20	Αςςυραςψ	93.01 %	99.68 %	92.97 %	99.66 %	92.98 %	99.66 %
ResNet56	Αςςυραςψ	93.29 %	99.75 %	93.12 %	99.76 %	93.21 %	99.65 %
MobileNetV2	Αςςυραςψ	93.31 %	99.44 %	93.15 %	99.45 %	93.08 %	99.39 %

Πίνακας 1.24: Ακρίβεια μοντέλων στο Versal, FashionMNIST

		FashionMNIST Απόδοση					
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
ResNet20	Καθυστέρηση	40.37 υς	23.09 υς	20.26 υς	18.54 υς	19.44 υς	21.50 υς
	Μέσα FPS	24 770	43 293	49 351	53 936	51 425	46 491
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
ResNet56	Καθυστέρηση	55.89 υς	31.37 υς	31.34 υς	31.20 υς	31.27 υς	31.18 υς
	Μέσα FPS	17 891	31 869	31 900	32 050	31 979	32 065
		1 Τηρεαδ	2 Τηρεαδς	3 Τηρεαδς	4 Τηρεαδς	5 Τηρεαδς	6 Τηρεαδς
MobileNetV2	Καθυστέρηση	101.94 υς	77.12 υς	77.12 υς	77.03 υς	77.02 υς	77.03 υς
	Μέσα FPS	9 809	12 965	12 966	12 982	12 982	12 981

Πίνακας 1.25: Απόδοση μοντέλων στο Versal, FashionMNIST

Με την εφαρμογή των μοντέλων στην πλατφόρμα μας επιτεύχθηκε η ενίσχυση της ακρίβειάς τους, ωστόσο δεν γνωρίζουμε την κατάσταση της ανθεκτικότητας. Καθώς δεν υποστηρίζεται η παραγωγή παραδειγμάτων στην πλατφόρμα, χρησιμοποιούμε τα ήδη παραγόμενα συνολα που χρησιμοποιήθηκαν για του προηγούμενους υπολογισμούς. Αναμένουμε να έχουμε υψηλά ποσοστά για τα προηγούμενα συνολα δεδομένων και μηδενικά στα τελικά. Εξετάζοντας τους πίνακες 1.26 έως 1.29 παρατηρούμε ορισμένα ενδιαφέρον αποτελέσματα.

		CIFAR-10			
		test 0	test 1	test 2	test 3
ResNet20	Κβαντισμένο Μοντέλο	90.75, 99.75	90, 99.75	87, 99.75	71.75, 99.75
	Versal	90.22, 99.75	89.22, 99.25	89.71, 99.75	73.43, 99.75
ResNet56	Κβαντισμένο Μοντέλο	93.25, 99.75	92.25, 99.75	90.75, 99.75	75.25, 99.75
	Versal	92.23, 100	90.98, 99.75	89.47, 99.75	72.9, 100
MobileNetV2	Κβαντισμένο Μοντέλο	92.25, 100	92.5, 100	89.75, 99.5	76.5, 99.25
	Versal	92.98, 100	91.72, 100	91.47, 99.74	76.19, 99.24

Πίνακας 1.26: PSNR μέτρηση μέσω ακρίβειας - Test Set

		CIFAR-10			
		train 0	train 1	train 2	train 3
ResNet20	Κβαντισμένο Μοντέλο	100, 100	99.75, 100	99.5, 100	85, 100
	Versal	100,100	99.49, 100	98.74, 100	83.70, 100
ResNet56	Κβαντισμένο Μοντέλο	100, 100	100, 100	99.25, 100	80.5, 100
	Versal	99.75, 100	99.5, 100	97.24, 100	79.20, 100
MobileNetV2	Κβαντισμένο Μοντέλο	100, 100	100, 100	99.25, 100	46,75.52
	Versal	100, 100	100, 100	99.74, 100	45.61, 75.68

Πίνακας 1.27: PSNR μέτρηση μέσω ακρίβειας - Train Set

		FashionMNIST			
		test 0	test 1	test 2	test 3
ResNet20	Κβαντισμένο Μοντέλο	92.5, 99.75	93, 99.75	92, 99.75	5.5, 99.75
	Versal	92.73, 99.75	92.48, 99.75	92.48, 99.75	74.18, 99.75
ResNet56	Κβαντισμένο Μοντέλο	94.75, 99.75	94.25, 99.75	93.75, 99.5	4, 100
	Versal	94.48, 99.75	93.98, 99.75	93.48, 99.5	85.21, 100
MobileNetV2	Κβαντισμένο Μοντέλο	93, 99.25	92.75, 99.75	93.75, 99.5	3.75, 99.75
	Versal	92.73, 99.25	93.23, 99.5	92.98, 99.5	87.72, 99.8

Πίνακας 1.28: PSNR μέτρηση μέσω ακρίβειας - Test Set

Αρχικά, για το CIFAR-10 στα τελευταία σύνολα όπου αναμέναμε μηδενικά αποτελέσματα, παρατηρούμε αυξημένες αποδόσεις. Το φαινόμενο αυτό οφείλεται στην απαιτούμενη κανονικοποίηση των δεδομένων. Η κανονικοποίηση κατατάσσεται στις μεθόδους προ-επεξεργασίας επεξεργάζοντας τα δεδομένα πριν εισαχθούν στο μοντέλο. Με τον τρόπο αυτό, οι διαταραχές κανονικοποιούνται μαζί με τα υπόλοιπα δεδομένα βάσει των τιμών

		FashionMNIST			
		train 0	train 1	train 2	train 3
ResNet20	Κβαντισμένο Μοντέλο	99, 100	99, 100	99.75, 100	0.75, 100
	Versal	99.5,100	99.5, 100	99.75, 100	79.95, 100
ResNet56	Κβαντισμένο Μοντέλο	99.75, 100	99.5, 100	99.75, 100	0.25, 100
	Versal	99.75, 100	99.75, 100	99.75, 100	87.47, 100
MobileNetV2	Κβαντισμένο Μοντέλο	99.75, 100	99.75, 100	99.75, 100	0, 100
	Versal	100, 100	100, 100	100, 100	95.74, 100

Πίνακας 1.29: PSNR μέτρηση μέσω ακρίβειας - Train Set

του CIFAR-10. Επομένως, με την χρήση προ-επεξεργασίας επιτυγχάνεται η ενίσχυση της ανθεκτικότητας των δικτύων τόσο στα εφαρμοσμένα στην πλατφόρμα μοντέλα όσο και στην κβαντισμένη έκδοσή τους. Τέλος, συγκρίνοντας τα αποτελέσματα για το FashionMNIST παρατηρούμε τα αναμενόμενα αποτελέσματα που αναφέραμε. Καθώς το σύνολο δεν απαιτεί κάποια επεξεργασία, παρατηρούμε μηδενική ακρίβεια στα τελικά παραγόμενα σύνολα για τα κβαντισμένα μοντέλα. Παράλληλα, παρατηρούμε αυξημένες ακρίβειες ακόμα και στα τελικά σύνολα, στα μοντέλα εφαρμοσμένα στην συσκευή μας. Το γεγονός αυτό υποδεικνύει την αυξημένη ανθεκτικότητα των μοντέλων χωρις βέβαια να μπορέσουμε να εκτιμήσουμε κατα πόσο. Εν τέλη, θεωρώντας οτι η ανθεκτικότητα έχει παραμείνει στα ίδια ποσοστά με τα κβαντισμένα μοντέλα, επιτυγχάνυμε την μηδαμινή πτώση της ακρίβειας με σημαντικές αυξήσεις ως προς την ανθεκτικότητα των δικτύων μεχρι και >40%.

1.5 Συμπεράσματα - Μελλοντικές Εξελίξεις

Συνοψίζοντας, προτείναμε μια μέθοδο ενίσχυσης της ανθεκτικότητας των νευρωνικών δικτύων για εφαρμογή σε ειδικού σκοπού συσκευές περιορισμένων πόρων. Με την μέθοδο αυτή, εκμεταλλευόμαστε την αποδοτικότητα των επιθέσεων κλειστού τύπου, παράγοντας παραδείγματα τα οποία χρησιμοποιούμε σε μια σειρά τριών επανεκπαιδεύσεων έπειτα απο την οποία κβαντίζουμε το δίκτυο. Με τον τρόπο αυτό. επιτυγχάνουμε την βελτιστοποίηση των δικτύων, εφαρμοζόμενα σε συσκευές περιορισμένων πόρων, διατηρώντας την ακρίβεια και ενισχύοντας την ανθεκτικότητά τους. Μέσω αναλυτικού πειραματισμού με τα δίκτυα ResNet20, ResNet56 και MobileNetV2 στα σύνολα δεδομένων CIFAR-10 και FashionMNIST, επιτυγχάνουμε πτώσεις ακρίβειας <2%. Επιπλέον, με χρήση του PSNR, επιτυγχάνουμε μειώσεις 42% και 37% για το ResNet20, 43% και 39% για το ResNet56 και 45% για το MobileNetV2 στα CIFAR-10 και FashionMNIST αντίστοιχα, με τα τελικά PSNR να κυμαίνονται μεταξύ 29-31 dB για το CIFAR-10 και 24-29 dB στο FashionMNIST. Όπως αναφέραμε και στο αντίστοιχο τμήμα, η πτώση του PSNR σημαίνει αυξημένη ύπαρξη θορύβου εντός των παραδειγματάτων που επιτυχώς παραπλανούν τα δίκτυα. Επομένως, η μεγαλύτερη πτώση μεταφράζεται σε αυξημένη ανθεκτικότητα κατα των επιθέσων. Παράλληλα, εξετάζοντας τα τελικά παραδείγματα, αντιλαμβανόμαστε οτι αυτά τα εύρη τιμών PSNR εκφράζουν την σημαντική ύπαρξη θορύβου σε σημείο εντοπισμού απο ανθρώπους, επιτυγχάνοντας έτσι τον σκοπό αυτής της εργασίας.

Μελλοντικές Εξελίξεις

Η εργασιά αυτή, ενώ παρέχει πολλά υποσχόμενα αποτελέσματα, μπορεί να βελτιστοποιηθεί περεταίρω ως προς την αποδοτικότητα και επεκτασιμότητά της. Μέσω αυτών των βελτιώσεων στοχεύουμε στην παροχή μιας ακόμα πιο ισχυρής και επεκτάσιμης λύσης για όλες τις ετερογενείς συσκευές.

- Βελτίωση ανθεκτικότητας με χρήση αλγορίθμων προ-επεξεργασίας και ειδκές τεχνικές.
- Χρήση, αντίστοιχου του HopSkipJumpAttack, αλγορίθμου επίθεσης ελαττώνοντας τις χρονικές απαιτήσεις.
- Ενσωμάτωση αποδεδειγμένων μεθόδων προστασίας συσκευών για την παροχή ολικής ασφάλειας.
- Επέκταση σε άλλες αρχιτεκτονικές αναγνώρισης εικόνων.

Chapter 2

Introduction

2.1 Introduction

Artificial Intelligence (AI) has gained significant attention in recent years, pervading various aspects of our modern lives. From virtual assistants on smartphones and home appliances, to autonomous vehicles and advanced medical diagnostics, these AI technologies driven by deep neural networks (DNNs) are increasingly intergrated into our daily routines. However, this rapid integration of such technologies exposes a critical issue: the vulnerability of neural networks to adversarial attacks. While neural networks have demonstrated incredible capabilities in terms of efficiency and accuracy, their susceptibility to adversarial attacks raises a serious concern over their reliability. Adversarial Attacks such as HopSkipJumpAttack, exploit vulnerabilities in the architecture of a neural network, leading to misleading and erroneous outputs. Furthermore the ever-increasing complexity of neural networks, coupled with the significant resources they require, intensifies these vulnerabilities, highlighting the need for extended research into their robustness. Despite the aforementioned achievements of Neural Networks, the adversarial robustness remains a crucial but underexplored area of research along with its implications for efficiency in various applications. In this thesis, we aim to address these critical challenges by investigating known strategies to enhance the robustness of neural networks against adversarial attacks. Our primary goal is to analyze state-of-the-art lightweight architectures such as ResNet20, ResNet56 and MobileNetv2 on CIFAR-10 and FashionMNIST datasets to assess the effectiveness of these strategies. Secondly, we aim to explore the impact hardware deployment has on a neural network model specifically utilizing the Versal Platform along with its VitisAI Toolset. This will help us simulate real-world applications, especially in edge computing environments where computational resources are limited.

Additionally, by evaluating our performance metrics, we can determine the viability of such strategies for edge deployment. To summarize, this thesis aims to bridge the gap between theoretical advancements in robustness research and their real-world applicability.

2.2 Thesis Structure

The thesis is organized in 5 chapters. Chapter [3] focuses on related work and research discussing existing defense strategies. Chapter [4] covers the required theoretical background and technologies that will be utilized for the reader. Chapter [5] details our defense techniques and the tools we use to apply them along with information for our Versal Platform and VitisAI toolser. Chapter [6] delivers our findings and results. Lastly, Chapter [7] provides an in-depth summary of our findings along with potential future studies and developments.

Chapter 3

Related Work

Development of adversarial defenses is a crucial but difficult process concerning multiple researchers. Gathering significant attention over the recent years, there are numerous studies on techniques aimed to mitigate the vulnerabilities of neural networks. The main issue however lies in the ever evolving Neural Network architectures and along with them, new vulnerabilities and attack mechanisms. While multiple defense methods have been proposed, each one aims to mitigate vulnerabilities exploited from specific attacks restricting their potential in generalization, rendering them obsolete when new attacks inevitably emerge. Moreover, there is minimal research for defense mechanisms in hardware solutions and how they might affect them, exposing a critical issue where edge-based neural network applications are fully exposed to such attacks. In papers [6, 29, 16], Gradient masking is proposed as a defense mechanism by attempting to hide the gradient of a network. This defense while effective against white-box attacks like FGSM, it involves further computational complexity while still remaining susceptible to other attacks that don't require knowledge of the gradient but rather estimate it. Furthermore, [58, 6, 29, 16] suggest a Defense Distillation technique which involves training a secondary version of the targeted model that has smoothened the decision boundaries rendering it less sensitive to minimal perturbations. While such a technique has been proven beneficial, it only affects white-box attacks and combined with its extensive computational requirements, its use in edge-based and resource-limited environments is at least problematic. Another effective technique however can be observed from [6, 16] suggesting an Adversarial Example detection mechanism through the use of another network such as GANs. With this method, a secondary binary classification network checks for perturbations in the input before the model evaluates it. Adversarial Example Detection however while proven effective against white-box attack generated perturbations, it requires an extensive computational overhead making its implementation impossible in edge and resource-limited applications. [58, 6, 29] also include Randomization as a defense strategy. With Randomization, small random perturbations are added in the training dataset during the initial model's training in order to minimize its sensitivity against adversarial examples. Utilizing this technique, it is possible enhance the robustness of a network however, it requires prior knowledge of the upcoming type of attack while it may irreversibly affect the neural network's pattern recognition. Another type of defense examined in [16], focuses on the transferability of adversarial examples. In order to implement this type of defense, the architecture of the model is altered by adding null labels effectively defending the model against transferable attacks. Transferability Blocking while not effective on its own since it doesn't protect the network from attacks targeting it, can be combined with other defenses to further enhance its robustness. Weight Pruning [61], is an other adversarial defense aiming to minimize the general sensitivity of a neural network. Similar to regular pruning, it targets the weights of the network reducing their datatype accuracy and removing less influential ones. This type of defense paired with Adversarial Training is proven to be an effective robustness enhancing technique however, the findings of [61], suggest high robustness and high accuracy can not be achieved at the same time. Adversarial Training is a technique examined in multiple research papers [58, 6, 29, 16, 61, 50] where adversarial examples are deliberately generated and are used as a retraining dataset on the model. This method is both proven and popular among adversarial defenses enhancing the robustness while retaining computational simplicity. Another effective technique is Data Preprocessing [29] where the input is modified through a selected processing algorithm before passed through the network. This way, minimal perturbations are smoothened from the processing algorithm resulting in correct predictions. Quantization is another important defense method evaluated by [27, 23, 50]. Quantization inherently reduces the datatype accuracy of the network from floating-point to integer arithmetic reducing its sensitivity to small perturbations in the data. It is a proven method of robustness enhancement by a large factor dependent on the type of quantization applied. In [27], Adversarial training along with quantization are utilized as a robustness improvement. However, due to the choice of 3-bit quantization, while a significant increase in robustness was observed, there was a similar decrease in the performance of the network. Focusing on hardware applications and deployment, [23] proposes defense methods on a hardware level against related attacks. While not related to Adversarial Attacks, these methods can be utilized in conjunction with other techniques for deployment in sensitive environments such as secure edge environments or Space-related Applications. Due to this uncertainty and the minimal information on adversarial robustness in hardware applications, we propose a combination of these techniques, most notably Adversarial Training, Preprocessing and Quantization, in order to develop robust hardware solutions capable of withstanding adversarial attacks up to the point of human detection.

Chapter 4

Theoretical Background

4.1 Artificial Neural Networks - Introduction

Artificial Neural Networks (ANNs), a fundamental block for artificial intelligence, aim to imitate the behavior and composition of biological neural networks. Much like biological neural networks built on neurons, artificial networks rely on their artificial neuron counterparts for their functionality. Each of these artificial neurons, perform a series of simple tasks based on their input. Specifically, the input is multiplied with a set of predetermined weights, the result of which is then summed to be used for the activation function. The activation function uses its given input to then produce an output essentially simulating an interaction with a biological neuron as closely as possible [12]. A visual representation of an artificial neuron and its process can be seen in fig.4.1 while their function can be seen in eq.(4.1.1). Moving forward, we will see that this process is the basic principle behind all types of neural networks, enabling them to learn and adapt to their input data. This also allows us to utilize multiple techniques and construct complex networks tailored to specific tasks and devices, leading to the numerous breakthroughs we observe today.

$$y(k) = F\left(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b\right) \tag{4.1.1}$$

Where:

- F is the transfer function.
- w_k is the weight value for a discrete time k with i going from 0 to m where m is the total weights.

- x_k is the input value for a discrete time k with i going from 0 to m.
- b is the bias of the specific neuron.
- y_k is the output value of the neuron for a discrete time k

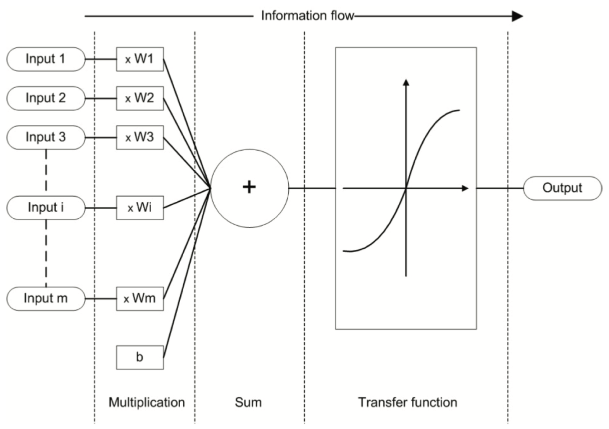


Figure 4.1: Principle of an Artificial Neuron, Source [12]

In the construction of neural networks, arranging artificial neurons into layers is fundamental step, with the amount of neurons per layer dependent on the chosen architecture. Integrating these layers within the input and output stages establishes our basic neural network. Furthermore, by incorporating multiple layers in our network, we gradually create more intricate architectures. To optimize and refine our networks based on each application, we are able to adjust each layer's characteristics. Notably, the size of each layer can be determined by the shape of the input data while the type is directly tied to the activation functions of the neurons. Typically we can choose from 6 different activation functions:

1. **Step Function**

   The Step activation function is a simple binary function where the result is compared to a predefined threshold. If the input value exceeds or matches the threshold, the output value is 1 while if the input value is below the threshold, the output value is 0. This can be also observed in the eq.(4.1.2).

$$y_1 = \begin{cases} 1, & \text{if } x_1 \cdot w_1 \geq threshold \\ 0, & \text{if } x_1 \cdot w_1 < threshold \end{cases} \tag{4.1.2}$$

2. **Linear Function**

   The Linear activation function calculates the input with $[F(x) = x]$ and forwards it to the result. Assuming the input is $x_i \cdot w_i$ and the function has a bias b, the output of the Linear function will be the result of $x_i \cdot w_i$ plus the bias value as shown in eq.(4.1.3).

$$F(x) = x \tag{4.1.3}$$

3. **Sigmoid (Non-Linear) Function**

   Entering the territory of more intricate non-linear functions, we have the sigmoidal function. Widely employed among various architectures, it produces values within the range of [0,1] as seen in eq.(4.1.4). This property, makes it ideal for deployment in output layers, especially for classification models.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \;\; F(x) \in [0, 1] \tag{4.1.4}$$

4. **Hyperbolic (tanh) Function**

   Similarly to the sigmoidal function, the hyperbolic activation function ranges its output from -1 to 1 as seen in eq.(4.1.5). However, in contrast to sigmoid, tanh is mainly utilized in the hidden layers of more complex neural networks.

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \;\; F(x) \in [-1, 1] \tag{4.1.5}$$

5. **Rectified Linear Unit (ReLU)**

   The Rectified Linear Unit (ReLU) function [28], while widely acclaimed for its use in Deep Neural Networks, is a simple yet highly effective function described in eq.(4.1.6). Compared to previous non-linear functions, ReLU maintains its effectiveness without any extensive computational overhead. However, in comparison to previous functions, it is possible for ReLUs to become ineffective or negatively impact an architecture's performance in cases of large input value fluctuations during training.

$$f(x) = max(0, x) \tag{4.1.6}$$

6. **Softmax**

   Lastly, Softmax (eq.(4.1.7)) is frequently found in output layers of multi-class classification models. This function is designed to convert the input scores from the network, into probabilities that sum up to 1 or 100%. This capability, renders Softmax ideal for applications in Image classification models where class scores can be interpreted to a confidence percentage.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{4.1.7}$$

4.1.1 Types of Neural Networks

While ANNs are regarded as the building block of multiple types of networks, we are focusing on a sub-category of them, the Deep Neural Networks (DNNs). DNNs are characterized from their numerous layers between their input and outputs compared to simplistic ANNs. Keeping characteristics of ANNs, DNNs are a more complex version of them providing higher sensitivity and a wider range of applications. In order to understand the diverse applications of DNNs, we briefly introduce some architectures after which we will delve deeper into Convolutional Neural Networks:

4.1.1.1 **Feed-Forward Neural Networks (FNNs)**

Feed-Forward Neural Networks define the basic architecture for deep and complex neural networks. Their distinctive attribute lies in facilitating a unidirectional flow of information, from the input to the output, restricting any interconnection with previous neurons or layers. By leveraging such linear structure, FNNs are capable to effectively learn the intricate correlations between the inputs and outputs, thus establishing an essential position in machine learning applications. A simple 2-layer FNN is shown on fig.4.2 While equations eq.(4.1.1.8), eq.(4.1.1.9) and eq.(4.1.1.10) describe the functions each layer's neurons.

$$n_1 = F_1\left(w_1x_1 + b_1\right) n_2 = F_2\left(w_2x_2 + b_2\right) n_3 = F_3\left(w_3x_3 + b_3\right) \tag{4.1.1.8}$$

$$m_1 = F_4\left(q_1n_1 + q_2n_2 + b_4\right) m_2 = F_5\left(q_3n_3 + q_4n_4 + b_5\right) \tag{4.1.1.9}$$

$$y = F_6 \left(r_1 m_1 + r_2 m_2 + b_6 \right) \tag{4.1.1.10}$$

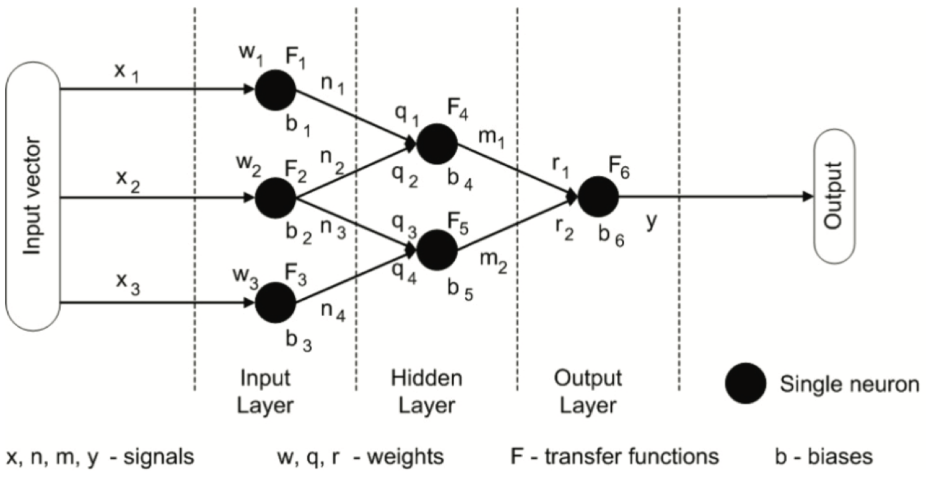


Figure 4.2: 2-Layer FNN example, Source [12]

4.1.1.2 **Recurrent Neural Networks (RNNs)**

Recurrent Neural Networks while similar to FNNs they have an crucial distinction in the flow of data. They permit the backward flow of information between neurons and layers. Such feature can be utilized by sharing data between layers but also by recurring the same information between neurons as seen in fig.4.3. Similarly to FNNs, RNN are utilized in various domains. However, their internal state and dynamic temporal behavior, make them sensitive to architectural designs and training methodologies.

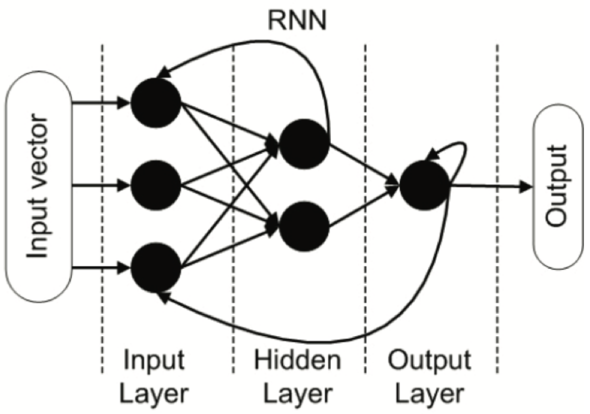


Figure 4.3: Simple 2-Layer RNN example, Source: [12]

4.1.1.3 **Long-Short Term Memory Networks (LSTMs)**

Long-Short Term Memory Networks are a specialized version of RNNs designed to recognize longer dependencies between data. As the name suggests, they overcome the limitations of RNNs in short data relations by offering special memory cells and mechanisms (as shown in fig.4.4) to support long-term associations with input and output data [49]. LSTMs have emerged as part of the most popular architectures in the recent years, finding widespread applications across sectors.

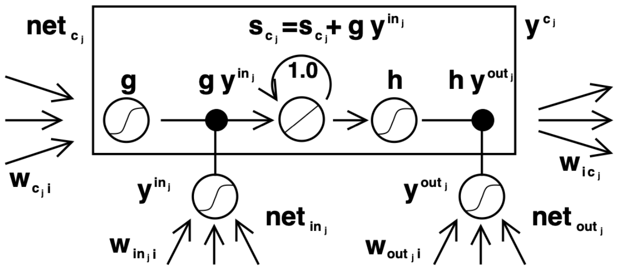


Figure 4.4: LSTM architecture of memory cell, Source: [49]

4.1.1.4 **Convolutional Neural Networks (CNNs)**

Convolutional Neural Networks are frequently utilized in image classification and pattern recognition tasks. While Convolutional Networks are fairly similar to conventional DNNs, their notable difference lies on the input type. CNNs are tailored and optimized for patter recognition in images with reduced parameters required compared to equivalent DNNs [35]. A regular 3-layer CNN is shown in fig.4.5 however we will discuss its architecture in greater detail in an upcoming section.

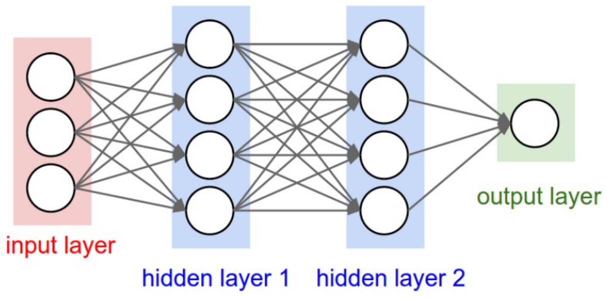


Figure 4.5: Example of 3 layer CNN, Source: [19]

4.1.1.5 **Generative Adversarial Networks**

Generative Adversarial Networks represent a distinct class in Neural Networks capable of generating diverse types of data. Their unique approach involves two separate neural networks: one for data generation and one for evaluation [32]. GANs have gained popularity over the recent years for their utilization in data augmentation, image synthesis and many more including vision related researches.

4.1.1.6 **Transformers**

Lastly, Transformers represent a shift in neural network architecture. Unlike previous types of models, Transformers rely on self-attention mechanisms to process sequential data. They prevail in tasks like Natural Language Processing (NLPs) due to their ability to capture long-term data dependencies without recurring connections as we saw with RNNs [14]. This unique characteristic of Transformers has lead to groundbreaking advancements in various fields, most notably in text generation with services like ChatGPT.

4.1.2 Training

Training is a crucial and resource intensive part of Neural Networks. While there are multiple training methods and techniques, it is beyond the scope of this thesis. As such, brief introductions to required concepts follow.

4.1.2.1 **Data Preprocessing**

Prior to training a Neural Network with our selected input, it is essential to ensure the data are scaled consistently across their dimensions. This is a critical step for a multitude of reasons but most notably to prevent the model favouring a subset of the data and maintaining its function's integrity. In order to process the data, we can perform two basic methods: Mean Subtraction and Normalization.

Mean Subtraction, a common form of preprocessing, involves subtracting the mean value from every dimension of the input data with the scope of centering it to 0. This is particularly helpful in Image Classification environments where the data except their size, often have 3 different color channels.

Normalization on the other hand, is the process of dividing the standard deviation of each corresponding dimension in the input data after they have been centered with Mean Subtraction. The result of the aforementioned processing is data centered around 0 with consistent values.

Assuming our input data are in an array x with d dimensions, the Mean Subtraction process is shown in eq.(4.1.2.11) and the Normalization process is shown in eq.(4.1.2.12). In eq.(4.1.2.13) we can see the full preprocessing operation on the data while in fig.4.6 we can observe the effect of Subtraction and Normalization.

Assume $mean[d]$ is a (d x 1) array where $mean[i]$ the mean value of $x[i]$ dimension.

$$x[i] = x[i] - mean[i] \tag{4.1.2.11}$$

Assume $std[d]$ is a (d x 1) array where $std[i]$ the standard deviation of $x[i]$ dimension.

$$x[i] = \frac{x[i]}{std[i]} \tag{4.1.2.12}$$

$$x[i] = \frac{x[i] - norm[i]}{std[i]} \tag{4.1.2.13}$$

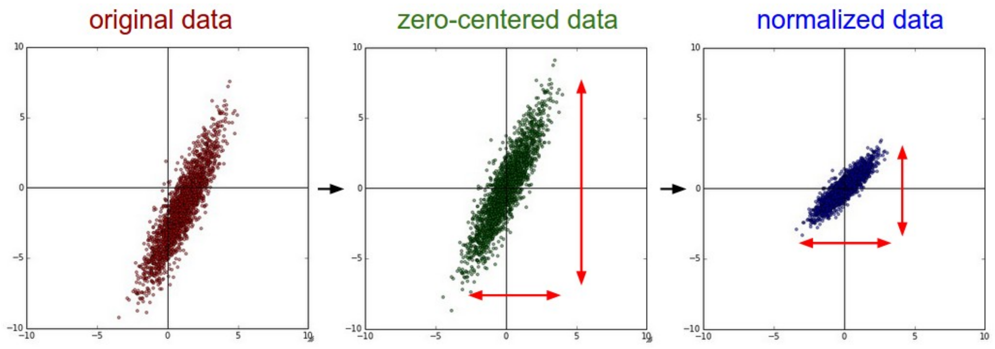


Figure 4.6: Normalization Process, Source: [19]

4.1.2.2 **Overfitting**

Another important part of training is controlling the data to prevent overfitting. Overfitting is the model's tendency to conform closely to training data failing to learn patterns and underlying behaviors. This results in poor performance and inability to generalize to new unseen data. Overfitting can occur either from a limited amount of training data or by prolonged training processes. Techniques like regularization, cross-validation along with early termination are utilized to prevent this behavior.

Regularization is a fundamental technique of overfitting prevention by penalizing the model's parameters. By imposing constraints during training, it discourages overly complex parameter values thus preventing overfitting and achieving an improved generalization performance. In order to better grasp how regularization works, let's examine two common types: L1 and L2 regularization.

L1 Regularization as a relatively common type, introduces a penalty λ , where λ the regularization strength, proportional to the absolute values of the model's parameters as seen in eq.(4.1.2.14). This type of regularization promotes data sparsity by enforcing values to approach zero leading to feature selection. While useful for complicated data with redundant or irrelevant information, L1 regularization may negatively impact a network's performance in detail-sensitive scenarios.

$$L_1 = \lambda \cdot |w| \tag{4.1.2.14}$$

L2 Regularization is the most common type of regularization. In contrast to L1, it introduces a penalty λ to the square magnitude of the model's parameters as shown in eq.(4.1.2.15). In practice, the $\frac{1}{2}$ term is often included for mathematical convenience, simplifying the gradient update to λw instead of $2\lambda w$. L2 regularization usually offers greater performance due to its nature targeting large abnormalities and values in data, preserving detail sensitivity and preventing overfitting.

$$L_2 = \frac{1}{2} \cdot \lambda \cdot w^2 \tag{4.1.2.15}$$

Cross-Validation is another form of overfitting prevention usually paired with early termination. Datasets employed in neural networks, are typically divided into two different sets: The training set, which is larger and its used for the training process and the test set, which is smaller and its used for evaluating the model's accuracy on unseen data. To implement cross-validation, the training set is split in 2 subsets: the new training set, which will be used for the model's accuracy and remains the larger of the two and the validation set, which will be integrated into the training process. During training, at the end of each epoch, the validation set is used to evaluate the model's performance thus simulating its final accuracy. As one can imagine, while a helpful method to assess the model's accuracy, it drastically increases the training process's duration and computational overhead thus it can be avoided in DNN applications.

Early Termination, as the name suggests, is a method of terminating the training process when a desired accuracy on the training or validation set is achieved. This method however, requires constant supervision in order to stop, rendering it inefficient in scenarios where large amounts of data and/or very deep neural networks are involved. In cases however where the desired accuracy is known beforehand, pre-coded thresholds can be utilized to leverage this method without supervision.

4.1.3 Optimizers

Optimizers are an essential part of neural networks, responsible for the model's effectiveness and performance during the training process. These algorithms are responsible for the parameter updates and the minimization of the loss function.

The **Loss Function** quantifies disparities between the outputs of the model and the ground truths of the labels in the training set. A lower loss signifies a stronger correlation between a predicted label and its ground truth meaning higher model confidence in its prediction. While multiple types of loss functions exist, they are not discussed further within the scope of this thesis.

Building on our understanding of the loss function, we can move to the optimization process assessing the types of optimizers and their hyperparameters.

Hyperparameters have a critical role in a neural network's optimization as they influence the behavior and performance of optimization algorithms. These parameters, defined before the training process, vary between optimizers and impact both the training and performance of the model.

- "Learning Rate" (lr), is a common hyperparameter existing in all optimizers defining the factor of which the training data affect the model. A high learning rate can lead to overfitting resulting in decreased accuracy while a low learning rate can affect the model's ability to convergence.
- "Batch Size" determines the samples used in each training iteration. While a larger batch sizes can lead to faster convergence and training, it requires larger memory. A smaller batch size however will lead to more updates thus prolonged training sessions and possibly slower convergence.

- "Momentum" is a hyperparameter found in SGD optimizer used to continue on previous gradient direction updates. This method accelerates the model's convergence while also smoothens the decent's trajectory.
- "Weight Decay" (L2 Regularization from section 4.1.2) adds a penalty factor preventing overfitting while also aiding the model to capture patterns within the training data.

Gradient Descent is a fundamental optimization algorithm adjusting the model's parameters by calculating the gradient of the loss function in relation to each parameter. After this computation, the algorithm updates each parameter in the direction resulting in reduced loss. This iterative process, controlled from a set of hyperparameters, guides the model to convergence and as such the model is trained.

Stochastic Gradient Descent (SGD)[7] is a variant of the Gradient Descent algorithm common in training processes. Unlike Gradient Descent, SGD calculates the gradient of the loss function using a subset of the training set known as a "mini-batch". This optimization technique allows faster and more efficient training on large datasets. Furthermore, SGD improves the generalization of the model by introducing randomness. However, this randomness also introduces noise which may lead to slower convergence and lower accuracies. For this reason, SGD also utilizes a momentum hyperparameter used to accelerate its convergence.

Adaptive Gradient Algorithm (Adagrad)[22] falls under the adaptive learning rate optimizers category utilized in multiple deep neural networks. The algorithm uses different per-parameter learning rates adjusted by the gradients observed through training. While practical for applications with sparse data, Adagrad may result in slow convergence due to its learning rate decay.

RMSProp[30] is a revised version of the Adagrad optimizer aimed to resolve the decaying learning rate issue. In contrast to Adagrad, RMSProp computes the moving average of squared gradients with a decay parameter and adjusts the learning rate accordingly. Due to this adjustment, RMSProp converges faster and is widely used as an efficient and effective optimizer.

Adam[36] is an evolved version of RMSProp utilizing both methods of Adagrad and RMSProp to adjust the learing rate. The adjustment results from both the exponential decay of the moving average of the gradient and the squared gradient leading to a smoother convergence. Along with the refined convergence, Adam also includes a bias correction to mitigate the effects of initial estimations. Its simplistic yet efficient design has made Adam a popular choice for DNN training aiding model generalization and optimization.

4.1.4 Quantization

Quantization is a crucial part of Neural Networks enabling further optimizations and deployment to hardware devices such as FPGAs and microcontrollers. Neural Networks as detailed above, are often designed in floating point arithmetic due to its high precision and accuracy. However, application specific and resource limited devices do not support this datatype and require conversion of the network into an integer type value in order to support it. Quantization is responsible for this conversion reducing the precision of the model's parameters and activation functions to the nearest supported quantization level. This reduction also decreases the computational complexity of the network leading to improved memory efficiency and lightweight models. Depending on the model architecture and the data distribution, it is crucial to pick the correct type and mode of quantization as it can lead to significantly reduced performance and accuracy. While quantization functions may differ from one another, a common choice utilized for both weight and activation functions is observed in eq.(4.1.4.16)[25].

$$Q(r) = Int(r/S) - Z \tag{4.1.4.16}$$

Where r is the real value input, S is the scaling factor and Z is an integer zero point.

This function essentially maps the real values r into their corresponding integer values in a method defined as "unified quantization" while the Scaling factor is defined as follows (4.1.4.17).

$$S = \frac{\beta - \alpha}{2^{\beta} - 1} \tag{4.1.4.17}$$

In this function the range $[\alpha,\ \beta]$ defines the clipping value range where the real values will be converted to while β denotes the quantization bit width. This clipping range is defined through the calibration process often automatic dependent on the framework utilized for the quantization. By assigning the minimum and the maximum of the real values as the clipping range, $[r_{\min},\ r_{\max}]$, we achieve an "asymmetric quantization" scheme since there are no guarantees that $r_{\min} = -r_{\max}$. This can be beneficial as asymmetric quantization results in a tighter clipping range useful in imbalanced weights and activations. In "symmetric quantization" however, these equations are simplified as seen in (4.1.4.18) and (4.1.4.19).

$$Q(r) = Int(r/S) \tag{4.1.4.18}$$

The integer zero point Z $= 0$

$$S = \frac{2max(|r|)}{2^n - 1}$$
$$\text{for full range quantization}$$
$$S = \frac{max(|r|)}{2^{n-1} - 1} \tag{4.1.4.19}$$
$$\text{for restricted range}$$

Full range quantization utilizes the whole integer space, e.g. INT8 is [-128,127] while restricted range is [-127,127].

Symmetric Quantization is generally a preferable option for weight quantization due to its simplicity and reduction in computational costs during inference. Another distinction between quantization types is found between dynamic and static quantization. Dynamic quantization calculates the aforementioned clipping range dynamically for each activation map during runtime. While this often achieves higher accuracy, it requires real-time calculations which is extremely computationally expensive for the target devices. Instead, static quantization can be utilized where the clipping range is pre-calculated during inference. This approach while not adding any computational overhead, typically results in lower accuracy[25].

4.1.4.1 Quantization Aware Training

During the quantization process, perturbations in the trained model's parameters can be introduced shifting it way from its convergence point reached with floating point precision. In oder to address this issue, we can retrain the Neural Network model with quantized parameters in order to converge to a better point with lower loss. Quantization Aware Training (QAT) achieves that by allowing the forward and backward pass of the quantized model to occur in floating point precision while the model parameters are quantized after each gradient update. Quantizing the parameters after the gradient update is the most critical step in this process since performing the backward pass by accumulating gradients in quantized precision can lead to error-prone or zero gradients. A schematic of the Quantization Aware Training process can be seen in fig.4.7.

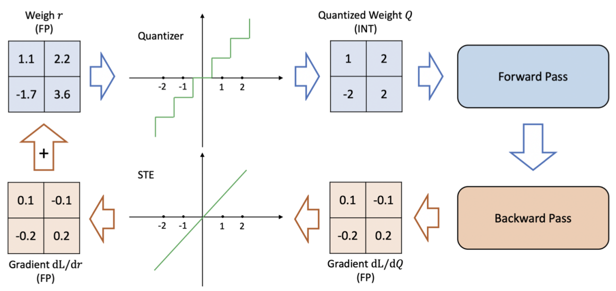


Figure 4.7: Quantization Aware Training Procedure, Source: [25]

While QAT is a proven working quantization method, its major disadvantage is the computational cost of retraining the model. This process can take multiple hundreds of epochs in order to recover to the accuracy while use of accelerators like GPUs is unsupported, further prolonging the process.

4.1.4.2 **Post-Training Quantization**

Post-Training Quantization (PTQ) on the other hand, is an alternative process performing quantization and adjustments on the weights without any fine-tuning. In contrast to QAT, it doesn't require the training data and comes with no computational overhead. PTQ only requires a small amount of data to use in the calibration process rendering it a preferred quantization method in situations with limited or unlabeled data. Along with these advantages however, PTQ often results in a lower accuracy of the model compared to its floating-point and QAT counterparts.

In fig.4.8 we can observe the differences between Quantization Aware Training and Post-Training Quantization. In cases where quantized models are deployed and utilized for prolonged periods of time, it is recommended to apply QAT in order to preserve the accuracy and details of the model. Whereas in multiple or short-lived deployments of models such as testing or research purposes, PTQ is a preferable process due to its efficient and quick application[25].

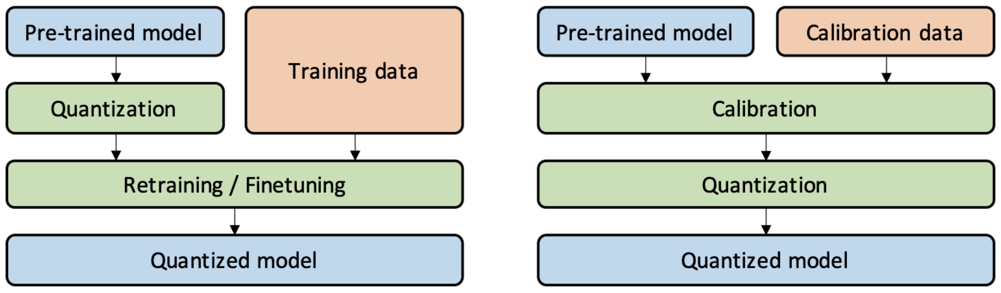


Figure 4.8: Quantization Aware Training (Left) - Post-Training Quantization (Right), Source: [25]

4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), a specialized subset of Neural Networks, are specifically refined for image data processing while retaining key features from conventional networks. By leveraging convolutional operations, CNNs are capable of handling the spacial properties of images and effectively capture their features. This specialization, has made CNNs ideal for image classification and object detection applications.

Image data processing differs significantly from other forms of data processing and can be both computationally expensive and time consuming. In order for a network to process an image, it must first be converted into an array of the pixel values. For colored images, this array is comprised of 3 dimensions: its width, height and color channels. For instance a small colored image from the CIFAR-10 dataset[38] of 32x32 resolution, will result in a 3 dimensional array of 32x32 data values for each color channel as seen in (4.2.0.20).

$$x[w][h][c] \text{ where w, h the width and height of the image in pixels and c the color channels} \tag{4.2.0.20}$$

In this representation, one very small image has 3072 values. Given that datasets contain images of higher resolutions and quantities, it is critical for a network to efficiently handle such data.

Traditional Neural Networks with fully-connected layers, in the case of CIFAR-10, would require 3072 weights only on their first layer rendering them impractical and impossible to scale to larger datasets. A CNN however, leverages the image's spacial structure by arranging the neurons in a similar 3-dimensional way for the width, height and depth of the layer. Furthermore, CNNs utilize convolutional layers that aren't fully connected, further enhancing their efficiency. Lastly, for image classification tasks, the resulting CNN's output layer consists of a single score vector providing confidence scores for each label. An example of such Convolutional Neural Network is depicted in fig.4.9

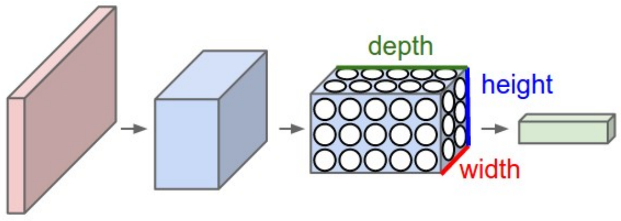


Figure 4.9: 3-Layer CNN Example: Input Layer -> Hidden Layer -> Hidden Layer -> Output Layer, Source: [19]

4.2.1 Building Blocks

Convolutional Networks Networks are comprised of a series of layers much like all other Network types. Contrary to other types however, most CNN architectures utilize 3 main types: a Convolutional Layer, A Pooling layer and a Fully connected layer.

4.2.1.1 Convolutional Layer (Conv)

Convolutional Layers, usually accompanied with a ReLU layer from section 4.1, consist of a predetermined set of filters. While each filter is smaller in regard to the input data resolution, it scans across the full depth of the input section-by-section, influencing its weights during the training process. This results in a per-filter sensitivity to different patterns, contributing to a general pattern recognition mechanism. These patterns can vary from specific color sensitivity to shape recognition, as such it is required to include multiple convolutional layers to achieve an accurate pattern prediction. Moreover, in order to avoid fully connected neurons in convolutional layers, neuron connections are established only in local regions adjusted by the "receptive field" hyperparameter. This connection however, occurs along the weight and height dimesions of neurons while the depth dimension remains fully connected. Due to this complexity of convolutional layers, more hyperparameters are utilized to adjust them:

- The **depth** hyperparameter defines the number of filters we will employ. This, along with the placement of each layer, directly impacts the details the network will capture from the training data. As we will see in different architectures, it is common practice to increase the filters as we approach the input layer in order to capture more details which can later be pruned if necessary.
- The **stride** parameter controls the filter's movement on the input images. Assigning the stride to 2 would slide the filter by 2 pixels each time across the images affecting the resulting volume of the output and sensitivities of the filters.
- **zero-padding** is a hyperparameter better utilized with size mismatches between the filter's and input's resolutions. It directly affects the output width and height of the layer by padding the input data with zeros across the borders allowing mismatched image-to-filter ratios to capture more data.

Along with the filter architecture, convolutional layers also utilize parameter sharing, allowing effective parameters to be reused in different spacial locations as an optimization technique, drastically decreasing parameter count. Lastly, ReLU layers placed in the 3-dimensional output of convolutional layers allow complex pattern recognition by adding a non-linear activation function.

4.2.1.2 Pooling Layer (Pool)

Pooling layers are particularly usefull for parameter reduction as they serve as down-sampling layers reducing the spatial dimensions of a convolutional layer's output. In CNN model architectures where multiple convolutional layers of various filter sizes are utilized, introducing a pooling layer in between, while not necessary, can aid with data retention and model optimization. These layers employ various pooling operations most notable of which is the max operation.

- **Max Pooling** selects the maximum value from the local regions in the output, disregarding less relevant information and preserving the most significant features. Due to its simplistic yet effective design, it remains as a common choice among other operations.
- **Average Pooling** is another operation where, instead of the maximum, the average value is used within a local region, capturing multiple features within that region. This results in a smoother down-sampling operation by sacrificing a small amount of detail.
- **L2-norm** on the other hand, computes the L2 (Euclidean) norm per local region, capturing a comprehensive amount of detail. While it may yield improved results compared to both max and average pooling operations, its computational overhead restricts its use-cases in DNN applications.

4.2.1.3 Fully Connected Layer (FC)

Finally, the Fully Connected Layer is a fundamental layer with complete connections with all activation functions of the previous layer. These types of layers are found in all types of Neural Networks while their activations are computed with a matrix multiplication followed by a bias offset. In CNN architectures however, fully connected layers are typically employed after the pooling ones, reducing the weights required for connections, leveraging both the detail and efficiency from previous layers.

With the foundational layer types in CNN model architecture defined, we can now analyze various model designs. Generally, CNNs follow a repeating pattern of the aforementioned layers. Examining the most common CNN architecture, we observe the repetition of Conv and ReLU layers along with optional pooling layers right after the input (4.2.1.21).

$$[\,[\,Conv -> \; ReLU\,] \,\cdot\, i -> \; Pool\,] \cdot j \tag{4.2.1.21}$$

Additionally, FC layers are paired with ReLU layers combining both the sensitivity of FC layers and the non-linearity of ReLU layers (4.2.1.22). Following multiple repetitions of this design, the output is processed through another FC layer that produces the class scores.

$$[\,FC -> ReLU\,] \cdot k \tag{4.2.1.22}$$

By utilizing such pattern, it is possible to create an adequately accurate and efficient Convolutional Neural Network model. For instance, assuming i = 3 for our Conv-ReLU pair and j = 1 for our cluster pair in eq.(4.2.1.21) and k = 1 in eq.(4.2.1.22), results in a CNN consisting of the following layers:

$$\begin{gathered} Input -> Conv -> ReLU -> Conv -> ReLU -> Conv \\ -> ReLU -> Pool -> FC -> ReLU -> FC \text{ (Output Layer)} \end{gathered} \tag{4.2.1.23}$$

4.2.2 Residual Networks Architecture

A major flaw of CNNs surfaces when multiple layers are introduced. Much like other DNNs, Convolutional Networks suffer from the common "Vanishing Gradient" issue. The Vanishing Gradient is a phenomenon present in Deep Neural Networks utilizing gradient based learning methods. During training, while the calculated gradient back-propagates from the output layer towards earlier ones, it slowly decreases its value reaching extremely low, close to 0, values exceeding the datatype's accuracy essentially vanishing. As a result, remaining layer weights are unable to update and the training process is significantly delayed or stopped.

Residual Networks (ResNets)[34], fix this phenomena by introducing new residual blocks capable of scaling to multiple layers while maintaining their light-weight and efficient design. The sole scope of these new blocks is to support the proposed method of *shortcut connections*. With this method, a layer is able to connect its activation functions to other layers skipping any in between as shown in fig.4.10, thus forming a residual block. This way instead of the stacked layers learning the underlying mapping of the network, by

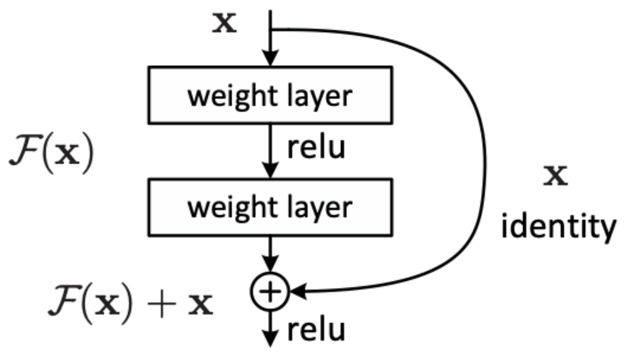

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.10: Example of shortcut connections, Source: [34]

introducing residual blocks, the layers are forced to fit in the residual mapping. Defining the initial mapping as $H(x)$, the new residual mapping can be defined as $F(x) := H(x) - x$ and as such the original mapping can be recast to $F(x) + x$. Compared to other CNN architectures, as we observe in fig.4.11, ResNets not only retain the gradient descent across their depth, but also enhance their accuracy by skipping under-performing layers during regularization. Moving forward for our experimentation, we utilize ResNet20 (A

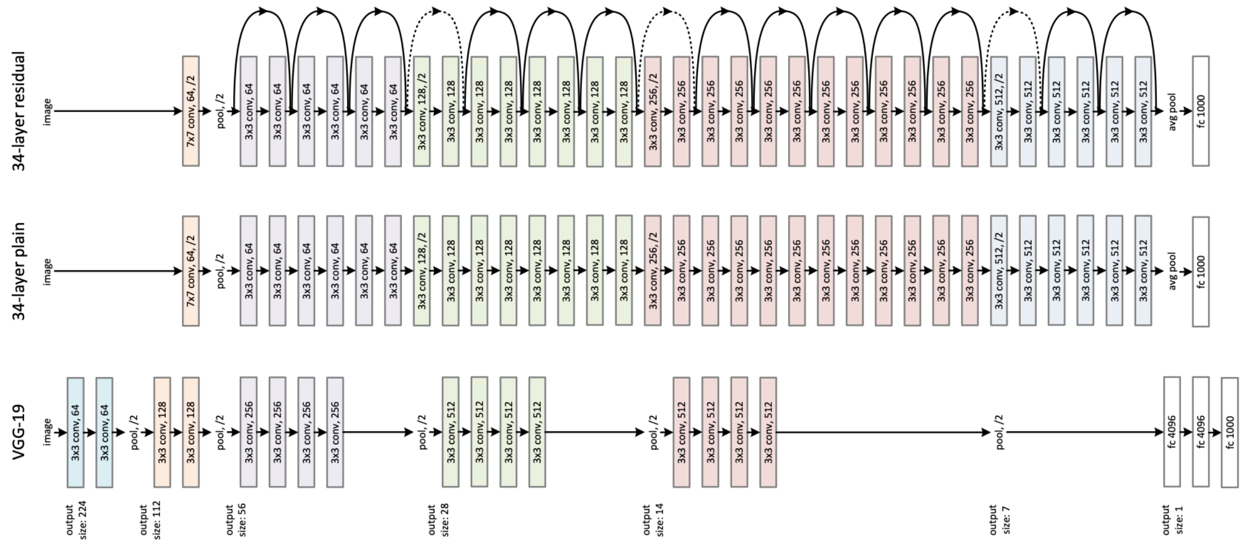


Figure 4.11: 34-layer ResNet model compared to VGG (bottom) and plain 34-layer CNN (middle), Source: [34]

20-layer Residual Network) and ResNet56 (A 56-layer Residual Network) Networks pre-trained on CIFAR-10 Dataset while we also train those models with the FashionMNIST Dataset [54]. These variants are picked for their efficiency and accuracy but also to emphasize how the depth difference of a network can affect its performance and robustness.

4.2.3 MobileNet Architecture

MobileNets [13], are an efficient class of convolutional neural networks tailored to mobile and embedded design vision applications. These lightweight networks result from a simplified architecture employing depthwise separable convolutions thus maintaining a balance between their speed, size and accuracy rendering them ideal for deployment in resource-limited devices. Their architecture consits of depthwise convolutions and pointwise convolutions working in conjunction with each other.

Conventional convolutions apply a convolutional kernel K across the entire depth of the input tensor. Assuming the input tensor has a shape of $w_i \times h_i \times d_i$ and the kernel has a shape of $k \times k \times d_i \times d_j$, (assuming a square kernel), the convolution's output will be of shape $w_i x h_i x c_j$. This in turn will have a computational cost of $w_i \cdot h_i \cdot d_i \cdot d_j \cdot k \cdot k$.

In contrast, a depthwise separable convolution is a two stage process drastically improving the computational cost. The first stage, a depthwise convolution, is applied on each input channel separately essentially filtering the input. Furthermore, in the second stage, a pointwise convolution is applied, where the outputs of the first stage are combined by utilizing a 1x1 convolution kernel across their depth essentially combining the features across channels. This separation achieves a computational cost of $w_i \cdot h_i \cdot d_i + d_i \cdot \left(k^2 + d_j\right)$ effectively reducing both the computational cost and the parameter count compared to traditional convolutions.

MobileNetV2 [44] is an evolvement of the MobileNet architecture leveraging the same principles with further improvements in performance. MobileNetV2 introduces the *Inverted Residual with Linear Bottleneck* block which handles all feature transformations minimizing computational complexity.
The **Linear Bottleneck layer**, crafted to enhance efficiency and performance, is a bottleneck layer responsible for feature transformations. This layer, comprised of three different operations, expands feature representations into a higher-dimensional space before compressing it back to a lower dimension.

- Expansion, where input features are expanded through a 1x1 kernel convolution (pointwise convolution) effectively enhancing their representational capability.
- Depthwise Convolution, where the expanded features are exposed to separate convolutional filters per channel, capturing spacial dependencies with a reduced computational overhead.
- Projecton, where finally a 1x1 convolutional projection layer transforms the features back to a lower dimension compressing the information while retaining any crucial characteristics.

With the utilization of such layer, MobileNetV2 achieves among performance enhancements, improved data retention and efficiency.

The **Inverted Residual Block** is a variation of the Residual Block as seen in 4.2.2. Retaining the shortcut connections of the Residual Block, it consists a Linear Bottleneck Layer aimed reduce computational complexity. In contrast to the Linear Bottleneck layer, the Inverted block first compresses the features before applying the convolutions after which the features are expanded. This inversion of the linear bottleneck process, along with the shortcut connections, effectively optimizes computational resources without sacrificing accuracy or efficiency.

While both MobileNet and MobileNetV2 are highly efficient architectures, MobileNetV2 has proven to be a more efficient and lightweight model for image classification purposes [21]. As such, along with Resnet20 and Resnet56, MobileNetV2 will be utilized in our experimentation with both CIFAR-10 and FashionMNIST.

4.3 Adversarial Attacks - Introduction

Adversarial Attacks, leverage vulnerabilities in Neural Network models to produce misleading inputs known as Adversarial examples. These attacks, modify the input data by introducing intentional imperceptible perturbations, invisible to the human eye, deceiving the model into misclassifying them. Earlier studies suggest the complex and non-linear architectures of DNNs is the root of these vulnerabilities due to their unpredictable behavior on minute changes in the input [52]. However, recent studies suggest the linear nature of both intricate and simpler Networks, is the underlying cause of such vulnerabilities with adversarial attacks exploiting the input's spacial characteristics [26].

4.3.1 Types of attacks

This manipulation of the input data while commonly referred to images, also affects multiple other data-types most notable of which are text and graph types.

4.3.1.1 Text-based Attacks

Text-based adversarial attacks generate perturbations to input text samples in order to deceive NLPs, RNNs and other transformer type models. Dependent on their resulting objective, these generated examples can be categorized in two types:

- **Sentence-Level Examples** retain the origin meaning of a sentence while misleading the model. These attacks employ methods like sentence rephrasing and paraphrasing along with context manipulation.
- **Word-Level Examples** on the other hand aim to deceive the model's comprehension of a word. These attacks utilize character insertion or deletion, commonly known misspelling, along with synonym substitutions.

4.3.1.2 Graph-based Attacks

Graph-based adversarial attacks, target Graph Neural Networks (GNNs) deliberately altering a graph's structure or node's features.

- **Structural Attacks**, involve modifying a graph's edges or nodes creating false relations or removing critical ones in order to disrupt the network's performance and classification.
- **Feature Attacks**, however, target the nodes of a graph, altering their attributes to closely resemble the original data leading to misclassification.

4.3.1.3 **Image-based Attacks**

Lastly Image-based adversarial attacks, generate imperceptible perturbations in images causing the network, usually CNNs, to misclassify them with high confidence scores. These types of attacks are categorized into two types:

- **Un-targeted attacks**, where the attacking algorithm's objective is for the model to misclassify the resulting input image with any label other than the ground truth.
- **Targeted attacks** [40], which require both the input image and the desired resulting class. These algorithms, generate perturbations in images with the sole scope of misclassifying them as the predefined resulting label. For instance, if an un-targeted attack successfully alters an image of a cat, the network may classify it as a car. A targeted attack however, where the resulting label is defined as a dog, will generate a similar example that the network should mispredict as a dog.

Regardless of the datatype they exploit, adversarial attacks are classified in 2 major categories, white-box and black-box attacks, dependent on the access to a model's architecture.

4.3.2 White-Box Attacks

White-box attacks require full knowledge and access to a model's parameters, gradient and architecture during both training and inference processes. With this type of access, attacking algorithms are able to exploit both the model's sensitivities and vulnerabilities generating effective adversarial examples. In order to further optimize this process, white-box attacks also utilize optimization techniques such as those discussed in section 4.1.3, reducing computational costs and time requirements while retaining their efficacy. Due to the nature of these types of attacks, they are usually adequately efficient and faster from black-box attacks as we will see later on. Some example white-box attack algorithms follow:

- HotFlip [24] is a text classification white-box attack aimed at character-level and word-level classfiers. The algorithm, based on character-level classification, utilizes the gradients of one-hot encoded input vectors in order to swap tokens with each other in an operation defined as "atomic flip operation".
- Graph attacks as proposed in [20] modify the structure data of a graph. It is important to note however that the paper proposes multiple types of attacks of both white and black box type.
- Fast Gradient Sign Method (FGSM) [26], is an image classification attack utilizing the gradient of the loss function in respect to the input data. Specifically, the FGSM algorithm, observed in eq.(4.3.2.24), evaluates the sign of the gradient from the loss function in respect to the input updating the perturbations according to the loss increase direction. An example of this process can be seen in fig.4.12.

$$adv_x = x + \epsilon \cdot \text{sign}\left(\nabla_x J(\theta, x, y)\right) \tag{4.3.2.24}$$

Where x the original image and ϵ the factor of the perturbation applied

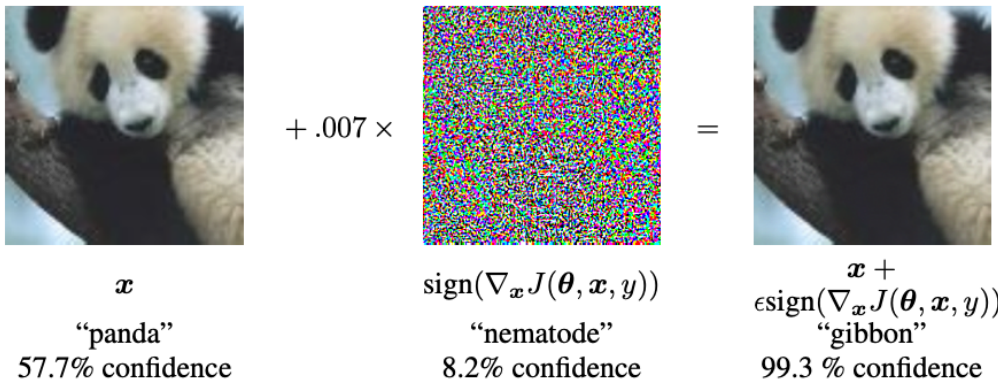

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 4.12: Demonstration of FGSM method, Source: [26]

- Lastly, Projected Gradient Descent (PGD) [43], another white-box image classification attack, works in an iterative manner. Building upon the basic iterative method, proposed as an enhancement of FGSM [40], PGD utilizes a 2 step process, in order to produce adversarial examples efficiently. The first step consists of identifying the input image before starting with a random perturbation around a specified radius. Iteratively, it follows the FGSM algorithm with a smaller factor by introducing a step size α as seen in eq.(4.3.2.25). The main advantage of PGD lies in the smoother creation of perturbations at the cost of increased computational complexity. An example of the algorithm's result can be seen in fig.4.13.

$$x_{t+1} = P\left(x_t \; + \; \alpha \cdot \text{sign}\left(\nabla_x J(\theta, x_t, y)\right)\right) \tag{4.3.2.25}$$

Where P defines the area of interest, limiting the perturbation's growth

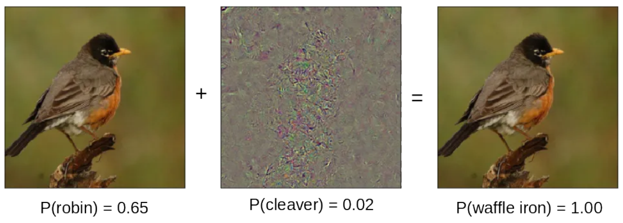


Figure 4.13: Demonstration of PGD method, Source: [37]

4.3.3 Black-Box Attacks

Black-Box attacks [48] on the other hand, do not require any knowledge of a model's architecture or parameters and rely on the labels and confidence output of the network in relation to the input data. These algorithms generate adversarial examples by iteratively querying the model and observing the corresponding outputs in relation to their input differences. While multiple black-box algorithms exist, a common approach is leveraging the transferability of a model. With this approach, adversarial examples can be crafted in a similar model architecture with efficient algorithms retaining their effectiveness when transferred back. This approach, while not always applicable, exploits the similar decision boundaries network architectures develop when trained for similar tasks enabling the generalization of adversarial examples. Due to the nature of operation of these algorithms, while inefficient compared to white-box attacks, they are preferred for real-world deployment evaluations as they mimic an attackers perspective and resources. Some examples of black-box attack algorithms follow:

- In the paper [8] a black-box algorithm for text-based classifiers is proposed utilizing genetic algorithms instead of gradient-based attacks. While image adversarial examples are indistinguishable from the human eye, altered sentences can be easily identified. Through the use of genetic algorithms, its proven possible to generate syntactically and semantically similar adversarial examples.
- Zeroth-Order Stochastic Gradient Descent Attack [15], is a optimization-based black-box attack simulating the SGD. Unlike SGD which requires access to the gradient of a models parameters in respect to the input data, the algorithm simulates the gradient but querying the model with small perturbations in the input observing the prediction changes. Utilizing this method, the algorithm iteratively updates the input image maximizing the loss function with respect to the model's predictions effectively generating adversarial examples.

4.3.4 HopSkipJumpAttack

HopSkipJumpAttack (HSJA)[17] is another query-efficient based black-box attack which we will utilize in this thesis for our adversarial examples and defense evaluations. HopSkipJumpAttack is proposed as a family of attacks capable of both targeted and untargeted attacks in an efficient and effective manner comparable to FGSM and PGD white-box attacks. The HSJA estimates a model's decision boundary through multiple queries with a magnitude of perturbations slowly approaching it. Utilizing a binary search, the perturbation magnitude is adjusted around the boundary, updating the sample image accordingly. Lastly, using smaller perturbations, the gradient is estimated around the boundary thus minimizing them and generating the

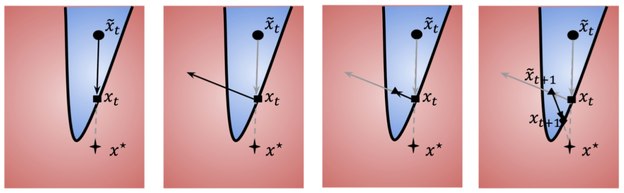


Figure 4.14: Intuitive HSJA explanation, Source: [17]

adversarial example. This process can be observed in fig.4.14 while the results for an untargeted attack on CIFAR-10 are seen in fig.4.15. In fig.4.16 we also observe similar adversarial examples for a targeted attack on CIFAR-10. The 1st column consists of the perturbations generated while the 2nd-9th column are the resulting adversarial examples based on the query limits of the model (100, 200, 500, 1K, 2K, 5K, 10K, 25K). The 10th column contains the original image.

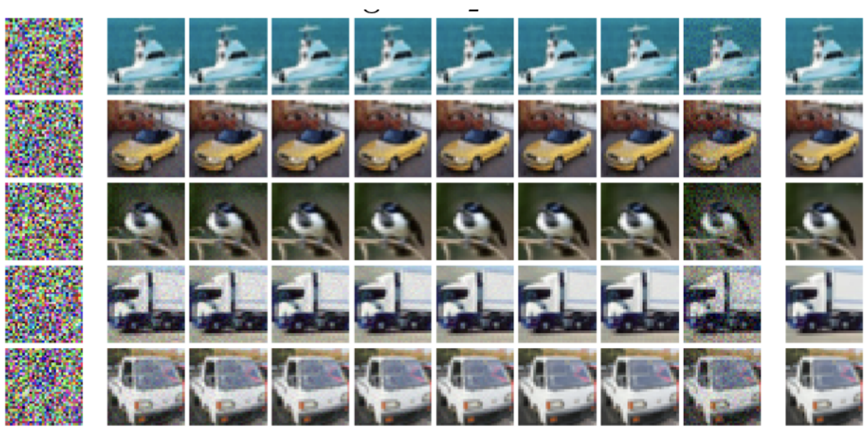


Figure 4.15: Untargeted Attack CIFAR-10, Source: [17]

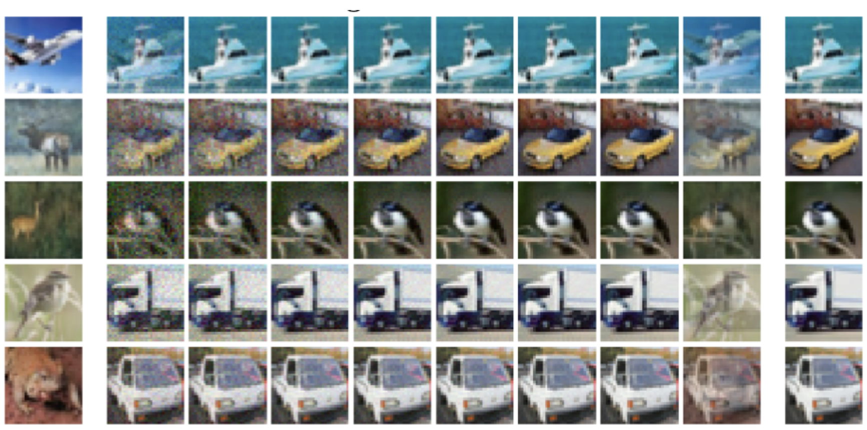


Figure 4.16: Targeted Attack CIFAR-10, Source: [17]

4.4 Adversarial Defenses - Introduction

Adversarial Robustness or a model's robustness to adversarial attacks is defined as the capability of a model to withstand adversarial examples correctly classifying them with their original labels. This term, in the from of "a model's robustness" will be utilized across this and upcoming sections.

Since Adversarial Attacks exploit different vulnerabilities of Neural Networks, researchers have developed various techniques aimed to effectively mitigate them. Ranging from adversarial training and quantization to gradient masking and defense distillation, multiple strategies have been proposed and utilized to enhance neural network robustness against adversarial manipulations. While some methods offer promising results in controlled environments, their scalability and applicability in real-world scenarios remain key challenges. Other methods on the other hand, offer effective mitigation tactics, applied only to specific attack types and inputs, rendering their generalization impossible. In this section we detail the most notable adversarial defenses, some of which we utilize in our implementation, along with their advantages and trade-offs.

4.4.1 Adversarial training

Adversarial training explored in [26, 31, 33, 39, 45], is the process of applying Adversarial Examples on the network as a robustness enhancement tactic. With this method, utilizing an attacking algorithm, the generated examples can be gathered into a new dataset on which the network can re-train on, building on its previous parameters. This way, the network preserves its accuracy on the initial dataset while modifying its decision boundary based on these examples. Adversarial training is a known method within the adversarial defense sector with multiple researches proving its effectiveness [51, 53]. However, the main issue of adversarial training lies in the chosen attack algorithm and the training process. In order to preserve the model's accuracy on the trained data during the re-training process, careful adjustments must be made on the hyperparameters along with extensive time investment heavily dependent on the complexity of the model and the length of the dataset. Even if the training process isn't time consuming, another important factor is the chosen algorithm. White-box attacks while efficient and fast, generate perturbations specifically on the neural network's weights. Training on these examples will undoubtedly enhance the robustness of the network but only for white-box related attacks leaving it exposed to real-world black-box related algorithms. On the other hand, utilizing black-box attacks may yield better results but requires significant time investment for the adversarial example generation. We chose this approach, applying the HopSkipJumpAttack algorithm, with similar efficiency to FGSM and PGD white-box algorithms, evaluating its effect on our networks.

4.4.2 Preprocessing

Preprocessing is another popular form of adversarial defense due to its simplistic yet effective nature in image classification networks. In essence, an adversarial example is an initial image with deliberately added noise in key positions. By preprocessing the image with an algorithm, these perturbations can either disappear or minimized to an extent where the model is unaffected. With various researches done on this topic, [41] suggests the use of a de-noising algorithm as a preprocessing step comparing the resulting image with the initial example. This achieves the removal of small perturbations while not only enhancing the robustness of the network but also providing and effective adversarial example detection method. Furthermore, other researches such as [59] suggest feature squeezing methods where the input image is modified, reducing its color-depth with pixel manipulation combined with spacial smoothing. Another preprocessing method is proposed in [60] where random pixels are removed from the image and a matrix estimation algorithm is utilized to reconstruct it. This way perturbations are smoothened out during the reconstruction process. While preprocessing is an effective defense tactic, more often than not it involves additional computational overhead directly affecting the performance of resource-limited devices. Moreover, it does not affect the model thus not enhancing its robustness but rather controlling its input. In this thesis, we investigate the effects of preprocessing, along with other utilized defense tactics, in the form of normalization. However, further research is required to assess its effects and benefits on non general purpose hardware devices.

4.4.3 Quantization

Quantization is a method of reducing the numerical precision of a neural network for deployment in resource-constrained devices. This reduction has gained the interest of researchers for possible implications with the model's robustness. Multiple publications, including [23, 42, 27, 50], prove that quantization while a necessary step in hardware deployment, can inherently increase the robustness of a network. Due to this numerical reduction depending on the quantization factor, the sensitivity of a neural network along with its accuracy are reduced, generalizing its decision boundaries rendering quantization an effective method against adversarial examples of any type. We utilize quantization not only as a mandatory step to deployment but also as a defense method evaluating its effects and results on our networks.

4.4.4 Gradient Masking

Gradient Masking is a form of an adversarial defense specific to white-box attacks. Initially proposed from [43], gradient masking aims to hide the gradient of the neural network from attacking algorithms exploiting it to generate examples. Proven effective against such attacks, it can be utilized to address this critical vulnerability of neural networks. However, the issue with gradient masking is unveiled through the use of gradient-estimating algorithms. Black-box attacks do not require any prior knowledge of the network except the targeted input and its predicting labels. Due to this function, gradient masking is not a viable defense method in restricted environments where the only interaction allowed with a model is through its input.

4.4.5 Defense Distillation

Finally, Defense Distillation is another form of white-box attack defense mechanism proposed by [47] and evaluated by [46]. Defense Distillation utilizes the Distillation method by training a secondary model. Distillation is a method of utilizing two neural networks where the first is a highly accurate and sensitive one and the second is a smaller more efficient network. Using the first model as the "Teacher" and the second as a "Student", the second model achieves better generalization compared to the first enhancing its robustness against adversarial examples. This method however, similarly to Gradient Masking protects only from white-box attacks while its extremely increased computational complexity restricts its usage in edge-based environments and resource-constrained devices.

4.5 Field Programmable Gate Arrays - Introduction

Field-Programmable Gate Arrays (FPGAs) are a family of semiconductor devices similar to Integrated Circuits (ICs) featuring programmable logic blocks and configurable interconnects. Unlike traditional Application-Specific Integrated Circuits (ASICs), FPGAs allow the dynamic configuration after their fabrication process. This capability paired with modern versions containing millions of logic cells allows users to implement a wide range of software algorithms and a tasks. While often compared to processors, FPGAs mimic the behavior of typical ICs rendering them an exceptional choice due to their cost-effectiveness and similar performance value.

4.5.1 FPGA structure

In terms of architecture, FPGAs consist of configurable logic blocks (CLBs), responsible for the execution of programmed operations, programmable interconnection wires, typically in a matrix structure and I/O (Input/Output) interfaces in order to transfer data to other devices fig.4.17. Furthermore, FPGAs include digital signal processing (DSP) blocks along with integrated memory to further improve their computational capabilities. All of these components are can be configured to implement highly complex functions and circuits.

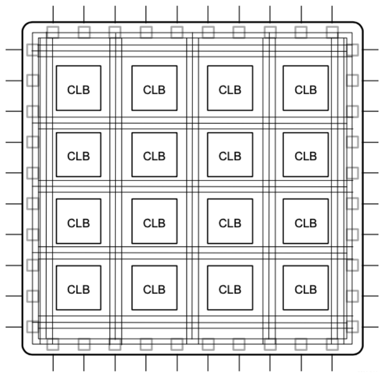


Figure 4.17: Simple example of FPGA Architecture, Source: [57]

The Configurable Logic Block (CLB) is a fundamental component FPGA architecture consisting of programmable look-up tables (LUTs), flip-flops (FFs) and multiplexers. LUTs store the logic tables of the programmed functions while the flip-flops act as storage elements within the block enabling sequential operations. Multiplexers route the various signals based on the input requirements of the programmed functions input. However, it is important to note that FPGA architectural implementations may vary between companies and manufacturers leading to differences in CLB designs.

Lookup Table units as previously mentioned, store the logic tables of functions working as truth tables. These tables are dynamically configured to implement various logic functions consisting of boolean values. As we observe in fig.4.18, the LUTs can be regarded as a collection of memory cells interconnected with multiplexers. Their input serves as selector bits on the multiplexers determining the corresponding output essentially working as both a computational unit and a storage element depending on the programed function[57].

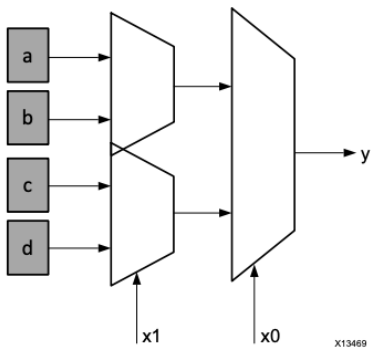


Figure 4.18: Representation of a LUT Source: [57]

Flip-flops on the other hand as seen in fig.4.19, always paired with LUTs, latch their input for one clock pulse after which its forwarded into the output pin. Featuring a clock-enable pin, the flip-flop is also capable of preserving its input for multiple clock pulses until that pin is reset[57].

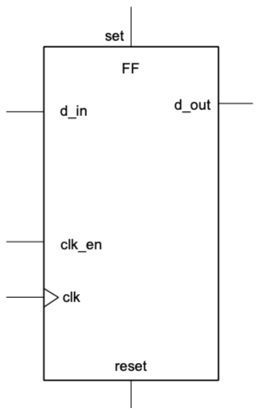


Figure 4.19: Representation of a Flip-Flop Source: [57]

DSPs, another crucial block in FPGA architecture, are arithmetic logic units (ALUs) composed of three smaller blocks. As portrayed in fig.4.20, these units consist of an addition/subtraction unit followed by a multiplier connected to an addition/subtraction/accumulate engine enabling them to compute complex functions in the form of (4.5.1.26) [57].

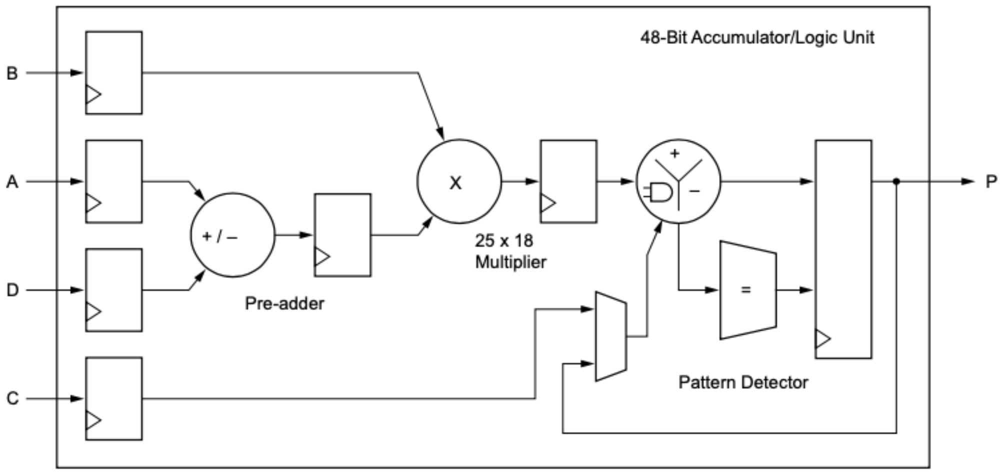


Figure 4.20: Representation of a DSP Source: [57]

$$\begin{aligned} p &= a \times (b + d) + c \\ p &\mathrel{+}= a \times (b + d) \end{aligned} \tag{4.5.1.26}$$

4.5.2 Versal Adaptive Compute Acceleration Platform

System-on-Chip (SoC) FPGAs are modern-type devices utilizing the benefits of both FPGAs and typical SoCs offering a versatile platform for diverse applications. With support for parallel heterogenous computing, these integrated circuits have lead the development of innovative and efficient devices across multiple computational domains.

One such family of devices is the **Versal Adaptive Compute Acceleration Platform (Versal ACAP)**™ which we will utilize for our neural network deployment. Specifically, we will deploy our models on the **Versal™ AI Core Series VCK190** [10].

Versal ACAPs integrate adaptable processing units and acceleration engines with programmable logic and connectivity enabling heterogenous hardware solutions for a wide range of applications fig.4.21. These devices incorporate Intelligent engines such as AI engines and DSPs, along with Adaptable and Scalar engines in an all-in-one integrated silicon package (Network-on-Chip (NoC)) offering exceptional performance-to-power ratios compared to conventional CPUs, GPUs and FPGAs. Particularly, ACAPs are comprised of a multi-core scalar processing system, an integrated block for PCIe® with DMA and Cache Coherent Interconnect Designs , SIMD VLIW AI Engine accelerators for artificial intelligence and complex signal processing along with Adaptable Engines in the programmable logic. This network also includes a management controller responsible for the configuration and booting sequence of each device. Furthermore, the Versal AI Core Series features medium density programmable logic along with AI and DSP acceleration engines highly optimized for functions in machine learning, convolutional neural networks and telecommunication applications. The AI engines are comprised of a 32-bit scalar RISC processor, fixed and floating point vector units, data memory, and interconnects allowing their use either as individual tiles, as a complete array or an assortment of the two[11].

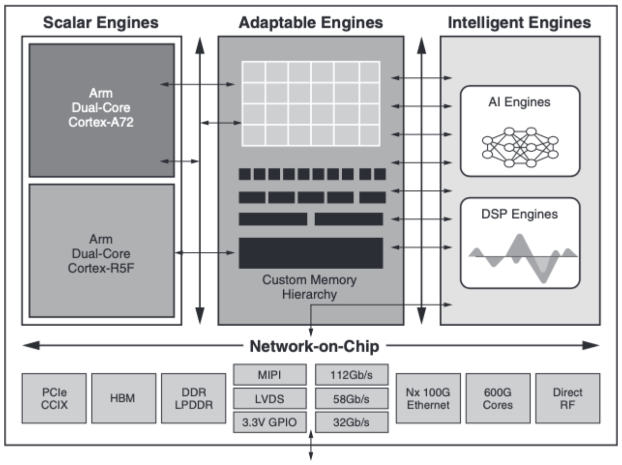


Figure 4.21: Xilinx Versal Diagram, Source: [9]

4.5.3 FPGAs and Neural Networks

FPGAs offer unique advantages in the AI domain. With their reconfiguration and parallelism, they allow customized hardware architectures enhancing neural network efficiency and optimizing resource utilization. By tailoring the hardware to the requirements of neural networks, FPGAs achieve greater performance-to-power ratios compared to general-purpose processors while their low-cost and low-latency with high-throughput capabilities make them ideal for edge deployment and real-time applications. Versal ACAPs build upon these advantages of FPGAs by integrating specially optimized hardware accelerators as previously detailed. With this heterogenous architecture they provide even greater efficiency and performance compared to other devices rendering them a great choice for neural network applications.

Chapter 5

Contribution

We propose a combination of sequential adversarial training along with quantization and deployment on a hardware platform as robustness enhancement technique assessing the effects on the models step by step. For our sequential adversarial training, we implement a three step process generating adversarial examples, retraining our models, evaluating the effects and repeating the process. After this method, we quantize the models evaluating their changes in accuracy and robustness before deploying them in our Versal ACAP AI Core Series VCK190 board further evaluating changes in performance.

5.1 Robustness Evaluation

Since we are on the realm of image classification, we must first define a reliable function to evaluate the variances between the original images in our datasets and the adversarial examples. For this usage we utilize the Peak Signal-to-Noise Ratio function (PSNR). This function is commonly used to measure the quality of compressed images in relation to their original versions calculating the error introduced from the compression algorithm in the image. This error is regarded as "noise" and is measured in decibels (dB). In essence, PSNR is an approximation metric of human perception of the reconstructed images rendering it an ideal choice for our use-case. This function, utilizes the mean square error (MSE) of the initial image (eq.(5.1.1)) in relation to its reconstructed counterpart as seen in eq.(5.1.2). As we observe however, the MSE accounts only for monochrome images featuring only one channel. In order to evaluate colored images we simply sum all the square differences per channel as seen in eq.(5.1.3).

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{5.1.1}$$

Where I is original image and K is the modified one.

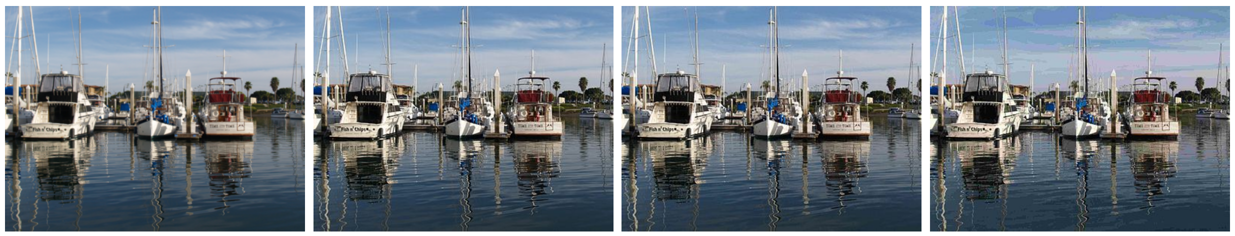
$$PSNR = 10 \cdot \log_{10}(\frac{MAX_I^2}{\sqrt{MSE}}) = 20 \cdot \log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{5.1.2}$$

Where MAX_I is the maximum pixel value of the original image

$$\begin{gathered} MSE_c = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I_c(i,j) - K_c(i,j)]^2 \\ MSE_{total} = MSE_R + MSE_G + MSE_B \end{gathered} \tag{5.1.3}$$

PSNR produces values in decibels corresponding to the error in the image. These values can range from ∞^*, where the images are identical and no noise is present, to $\approx$ 0dB where the resulting image has no relation to the initial one. This provides us with quantified results comparing the adversarial examples and evaluating our model's robustness. Furthermore, while PSNR is not a preferable choice for compression algorithm evaluations, we can see, comparing the differences in fig.5.1, that it allows us to estimate a PSNR value where human detection of adversarial examples is possible. We propose that values ranging from 29-24 dB are enough to visibly detect perturbations in the image.

*Infinity occurs from division by zero which we do not allow in our code restricting our maximum value to 100dB.



(a) Original Image (b) PSNR = 45.53dB (c) PSNR = 36.81dB (d) PSNR = 31.45dB

Figure 5.1: Example of PSNR for compression evaluation, Source:[3]

5.2 Setup

5.2.1 Models and Datasets

For our tests and evaluations we chose to work on ResNet20, ResNet56 and MobileNetV2 due to their lightweight and accurate nature. These models are supported on our Versal board minimizing any risk of conversion or deployment implications. Detailed in section 4.2, both ResNet and MobileNetV2 are proven efficient and accurate architectures while achieving a lightweight form factor capable of deployment in hardware devices. ResNet20 featuring 20-layers and 0.27 million parameters is a common choice as a simple "initial" model aiding us in running multiple tests and evaluations in order to optimize and perfect our codes and methods. Along with its time efficiency benefits, it also features high accuracies in small datasets which we compare to ResNet56 and MobileNetV2. ResNet56 features 56-layers and close to a million parameters rendering it a slower more complex model. This complexity however comes with increased accuracy and possibly increased robustness, a fact which we examine with this process. Finally, MobileNetV2 is utilized with a width multiplier of 1.4 featuring 4.33 million parameters. This model is chosen due its higher complexity and accuracy compared to the ResNet models but also for its optimized architecture for resource-limited device deployment. Furthermore, assessing a different model architecture with the same process can help us verify its effectiveness and scalability.

These models are trained and evaluated on CIFAR-10 and FashionMNIST datasets. CIFAR-10[38] is a dataset containing 60000 colored images of 32x32 resolution equally distributed in 10 classes. This dataset is a top-choice in the computer vision field due its small but effective nature featuring intricate images in a diverse set of classes. Essentially, it is a great choice for benchmarking different architectures, algorithms and techniques without requiring extensive resources and time compared to other large datasets. CIFAR-10 also requires a normalization process which can be regarded as a preprocessing technique in our application and will produce interesting results as we will see later on.

FashionMNIST[54] is the second dataset we utilize consisting of 70000 grayscale images of 28x28 resolution equally distributed in 10 classes. Building on the MNIST dataset which consisted of handwritten number images, FashionMNIST features multiple types of clothing offering a more complex and challenging dataset for neural networks. This dataset was chosen for its similarity in CIFAR-10 to further examine our process's efficacy on other diffenent datasets. Moreover, FashionMNIST requires no normalization enabling us to approximately evaluate the effects preprocessing by comparing the results from CIFAR-10 and FashionMNIST datasets.

It is worth noting however, that training these models on both datasets is unrelated to our objectives and requires extensive time investment and optimization of the hyperparameters. For this reason we utilize the available pre-trained models from [18] on CIFAR-10 while we do train our models on the FashionMNIST dataset since the process was rather quick yielding adequate results.

5.2.2 Tools and Services

Working on Neural Networks, Python is chosen as a preferred coding language due to its simplicity and efficiency along with its extensive support for them. Python has been in the forefront of Neural Network development and research in recent years offering a multitude of packages and frameworks to work with, thus establishing it as a main choice in the sector. For our neural network development we chose the PyTorch Framework[4] for its ease of use and feature-full capabilities. Additionally, we use the Adversarial Robustness Toolbox library available at [5] for its HopSkipJumpAttack support for our adversarial examples generation. The computational requirements of this thesis are covered through various cloud services like Kaggle [2] and Google Colaboratory [1] providing CPUs and GPUs through Jupyter notebooks.

5.3 Implementation

Before starting our adversarial defense methods, we first train all three models on the FashionMNIST dataset using the SGD optimizer, the importance of which will be observed later on, achieving top-1 accuracies of 92.24%, 93.72%, 93.72% for each model respectively. While these percentages can be considered relatively low in comparison to other pre-trained models, they are adequate enough for our utilization in this experiment.

First and foremost, in order to utilize the HopSkipJumpAttack for adversarial example generation, some variables must be set as limitations and initializations of the attack. It is also important to note moving forward that tensors, commonly used in neural networks, are not supported by the toolbox or the attacks and as such they are converted to numpy (a python library) arrays, with no loss in precision, when required.

- **classifier** defines the targeted model wrapped in a PyTorch classifier class, provided from the toolbox, containing its shape requirements, preprocessing steps and more.
- **batch_size** is the batch size defined by the estimator of the model during inference which in our case for all three models and two datasets is 64.
- **targeted** changes the mode of the attack between targeted and untargeted. We are only interested in untargeted attacks as targeted ones involve multiple implications and do not account for the whole

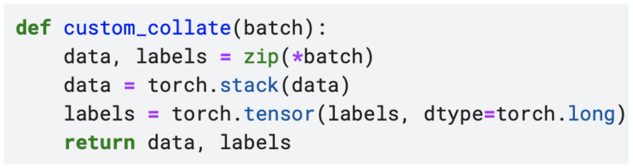
model's robustness.

- **norm** defines the norm used for the gradient estimation. Here choices can vary from infinite or 2. We use the default value of norm 2.
- **max_iter** defines the maximum number of iterations allowed over one singular image. We use the default value of 50 since lower iterations will negatively impact the effectiveness of the algorithm while larger values will dramatically increase its processing time.
- **max_eval** sets the maximum number of evaluations allowed for the gradient estimation. We also leave this in its default value of 10k. Changing this number affects the queries done on the model as seen from the examples in 4.3.4.
- **init_eval** sets the starting number of evaluations for the gradient estimation. Its default value of 100 remains unchanged during our experiments.
- **init_size** sets the maximum number of trials allowed for initial generation of adversarial examples. Remaining unchanged with the value of 100, this restricts the algorithm to a maximum of 100 random starting points for the examples.

Due to the randomness involved in neural networks let alone in adversarial attacks, in order to guarantee the reproducibility of our results, we use a seed of **0** in all of our coding environments.

For our initial adversarial training dataset we chose an adversarial example size of 30.000, accounting for 50% of the CIFAR-10 dataset and 42% of the FashionMNIST dataset. Since we intend to apply multiple rounds of adversarial training, we chose to initially expose the models to a slightly altered version of a large portion of their dataset in order to slowly adjust their decision boundaries preserving their accuracy. In upcoming iterations, we lower the amount of adversarial examples per dataset examining the generalization ability of the models in respect to their robustness. While it has been proposed to use a mix of both the initial training data and adversarial examples during adversarial training, we chose this approach to avoid overfitting and overexposure of the models to the same data since both of our datasets are relatively small.

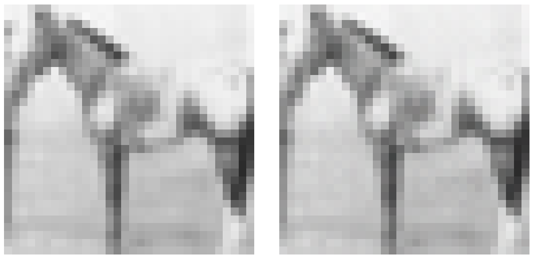
For our adversarial training process, we create new Tensor Datasets through built-in functions of PyTorch along with their DataLoaders. DataLoaders are responsible for the iteration and modification of the dataset. Their purpose ranges from applying required transformations on the data to splitting them according to the batch size and enabling parallelism. Here is where our first challenge is encountered. As previously mentioned, the toolbox and attack require the images in a numpy array form while PyTorch utilizes tensors. While we can transform our data between those two types with no issues, our dataloader requires explicit definition of the collate function fig.5.2 in order to iterate over them in our new dataset.



```python
def custom_collate(batch):
    data, labels = zip(*batch)
    data = torch.stack(data)
    labels = torch.tensor(labels, dtype=torch.long)
    return data, labels
```

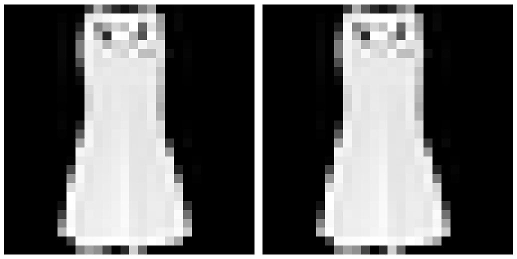
Figure 5.2: Custom collate function for data iteration

Examining our adversarial examples before starting the retrain process, we can see side by side the imperceptible differences between the original images and the examples in figures figs. 5.3 to 5.5 for CIFAR-10 and FashionMNIST along with their PSNR values. As we observe, while the PSNR values indicate there is a difference between them, we are unable to detect it. Our objective here is to reach a robustness level where the PSNR is low enough for these differences to be apparent.



Original - Horse 52.05 dB - Cat

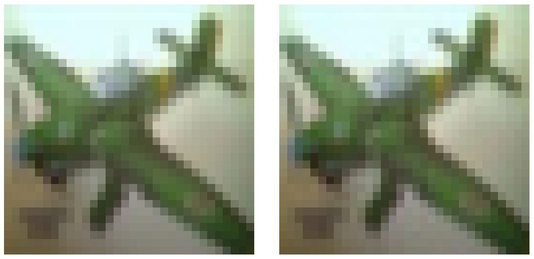
(a) CIFAR-10 Test Set example



Original - Dress 51.71 dB - Coat

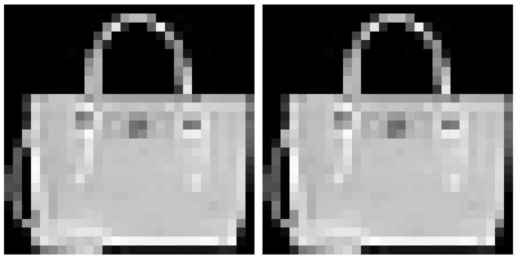
(b) FashionMNIST Test Set example

Figure 5.3: ResNet20 Adversarial Examples



Original - Plane 49.12 dB - Horse

(a) CIFAR-10 Test Set example

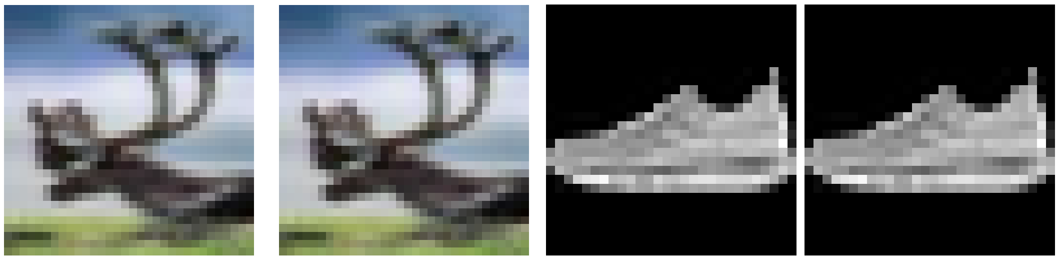


Original - Bag 39.73 dB - Shirt

(b) FashionMNIST Test Set example

Figure 5.4: ResNet56 Adversarial Examples

While the adversarial examples are generated from the training dataset, since they are used for training, we must generate new examples on the test set of the models in order to have an initial metric to assess their robustness. In our use-case since the robustness is evaluated in the form of PSNR, we can calculate the geometrical mean (eq.(5.3.4)) of all PSNRs from all the images in the adversarial test dataset.



Original - Deer 53.23 dB - Cat

(a) CIFAR-10 Test Set example

Original - Sneaker 44.33 dB - Shirt

(b) FashionMNIST Test Set example

Figure 5.5: MobileNetV2 Adversarial Examples

$$G_{mean} = \sqrt[n]{\prod_{i=1}^{n} x_i} \tag{5.3.4}$$

However, this requires the generation of an adversarial test dataset the size of which should be adequate to capture enough possible fluctuations. Through thorough testing and evaluation, we found that the difference in the geometrical mean of the PSNR in 400 test images compared to the geometrical mean of the PSNR in 30000 test images is within ± 0.005dB. Thus moving forward we accept this minute difference and evaluate the robustness of the models by generating a psnr-checking test dataset of 400 examples saving large amounts of time and resources.

In order to start our training process we use the SGD optimizer since it was used for the original training of our models. However, we must first adjust the learning rate, decreasing it by several orders of magnitude, in order to avoid overfitting on the adversarial examples and reduction of accuracy in our test set. Additionally, we implement a type of cross-validation technique where with each epoch we verify the accuracy on the adversarial set and the original test set in order to minimize precision losses and maximize the accuracy on the adversarial set. This allowed us to achieve an accuracy of 99+% on the adversarial dataset while losing under 0.5% on our model's accuracy. Finishing our first iteration of adversarial training, utilizing the PSNR function as mentioned above, we observe an incremental decrease in the PSNR as shown in table 5.1 verifying the effectiveness of adversarial training as an adversarial defense. In this step, it is important to note that for ResNet20 in FashionMNIST we observed an accuracy increase of 1.3% along with the robustness enhancement possibly signifying the potential of improvement in our initial training.

Moving to our second iteration of adversarial retraining we reproduce the steps as mentioned above, whereas in this iteration, our adversarial set is 20.000 samples accounting for 33% of the CIFAR-10 dataset and 28.5% of the FashionMNIST dataset. We made this change in order to reduce time consumption and

	CIFAR-10		**FashionMNIST**	
	Initial Dataset	Adversarial Training 1	Initial Dataset	Adversarial Training 1
ResNet20	56.68 dB	53.37 dB	48.67 dB	43.76 dB
ResNet56	54.91 dB	50.32 dB	47.28 dB	43.75 dB
MobileNetV2	58.39 dB	50.55 dB	45.78 dB	44.00 dB

Table 5.1: PSNR after Adversarial Training compared to initial

evaluate the ability of our models to generalize in respect to the adversarial examples further improving their robustness, as proposed by multiple publications on adversarial training which we detailed on the related section. Utilizing the same techniques as before, we are met with another challenge where the models are unable to train on the adversarial set or retain their accuracy to acceptable levels. To solve this issue we utilized a technique called "optimizer-switching" common among SGD optimizer trained models. With this technique, we change our optimizer to Adam in order to achieve greater performance. Compared to SGD, Adam achieves faster convergence and better generalization making it common practice to train models on SGD and switch to Adam for further improvements. Utilizing this method, we again achieve 98+% accuracy on the new adversarial dataset with relatively low percentage losses ranging from -0.5 to -0.8% with our robustness results in respect to the original data seen in table 5.2.

	CIFAR-10		**FashionMNIST**	
	Initial Dataset	Adversarial Training 2	Initial Dataset	Adversarial Training 2
ResNet20	56.68 dB	50.21 dB	48.67 dB	40.51 dB
ResNet56	54.91 dB	48.50 dB	47.28 dB	40.34 dB
MobileNetV2	58.39 dB	48.29 dB	45.78 dB	41.52 dB

Table 5.2: PSNR after second Adversarial Training compared to initial

Finally, for our third and final adversarial training iteration, we decrease our adversarial example set to 10.000 samples accounting for 16% of the CIFAR-10 dataset and 14% of the FashionMNIST Dataset. Leveraging the same generalization technique mentioned above, by reducing our dataset yet again we conserve both resources and time while further enhancing the robustness of our models. For our training process we again utilize the Adam optimizer achieving 98+% accuracy on our new adversarial examples with, this time, extended accuracy losses of $\approx<1\%$. Calculating the PSNR for all of our iterations we can observe the incremental robustness increase through tables 5.3 and 5.4 for CIFAR-10 and for FashionMNIST along with figs. 5.6

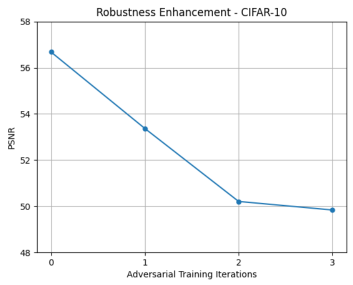
and 5.7 for a more intuitive approach.

	CIFAR-10			
	Initial Dataset	Adversarial Training 1	Adversarial Training 2	Adversarial Training 3
ResNet20	56.68 dB	53.37 dB	50.21 dB	49.84 dB**
ResNet56	54.91 dB	50.37 dB	48.50 dB	46.83 dB
MobileNetV2	58.39 dB	50.55 dB	48.29 dB	48.42 dB

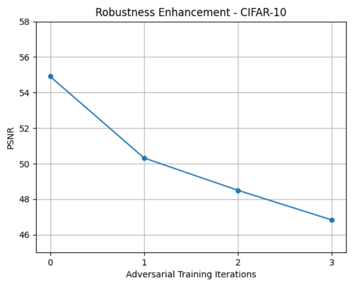
Table 5.3: Incremental PSNR decrease from Initial Model to 3 Adversarial Training procedures, CIFAR-10

	FashionMNIST			
	Initial Dataset	Adversarial Training 1	Adversarial Training 2	Adversarial Training 3
ResNet20	48.67 dB	43.76 dB	40.51 dB	37.76 dB
ResNet56	47.28 dB	43.75 dB	40.34 dB	37.36 dB
MobileNetV2	45.78 dB	44.00 dB	41.52 dB	38.98 dB

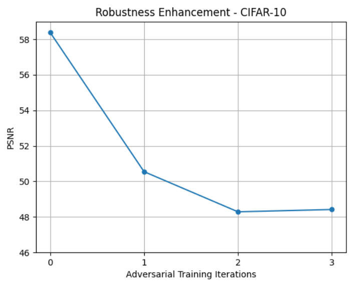
Table 5.4: Incremental PSNR decrease from Initial Model to 3 Adversarial Training procedures, FashionMNIST



(a) ResNet20 Model

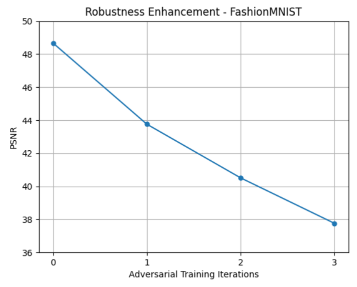


(b) ResNet56 Model

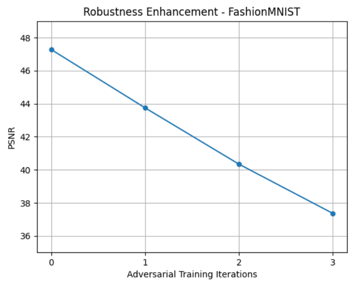


(c) MobileNetV2 Model

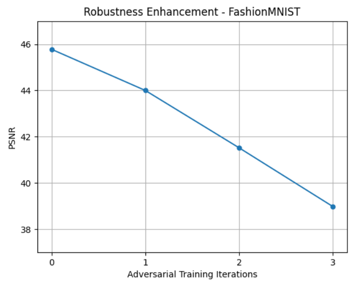
Figure 5.6: Incremental Robustness Enhancement - CIFAR-10



(a) ResNet20 Model



(b) ResNet56 Model



(c) MobileNetV2 Model

Figure 5.7: Incremental Robustness Enhancement - FashionMNIST

5.4 Quantization

We also utilize quantization as a robustness technique along with its use for our upcoming deployment in our device. As we have detailed in the adversarial defense section 4.4, quantization is a proven effective robustness technique with varying results dependent on the quantization level. While we want to evaluate the effectiveness of quantization to our models, we also want these results to be relatively comparable to our model's performance on the device. For this reason we chose Post-Training Static Quantization supported by PyTorch requiring only minor adjustments on our models. It may be possible to achieve better results through Quantization Aware Training however, it involves both additional computational and time investments along with the use of floating point variables during training. Thus in order to deploy these models, the quantization process would have to be rerun, quantizing the floating point variables, possibly negating any additional benefits of QAT. In PyTorch, in order to quantize a model using PTSQ, the model must be first wrapped in a Quantization Wrapper provided by the framework. Additionally, we must set our quantization configuration for our weights and activations. This configuration includes the type of quantization and the clipping ranges along with their quantization factor and quantizing function, attributes which we explained in section 4.1.4. Our quantization scheme consists of 8-bit quantization, supported by our hardware device, and full quantization clipping range along with a moving average min/max function for both the weights and activation functions. Furthermore, the quantization process requires a small part of the dataset for calibration purposes which we provide from the original datasets.

In order to evaluate the effects of quantization, since we are unable to train the models after they are quantized, we quantize the models in each step of the adversarial training process. Specifically, we quantize the initial model to achieve a base metric of both the effect of quantization and the adversarial training combined with it. Finishing each adversarial training iteration, we again quantize the model and generate new adversarial examples in order to further evaluate each step. Applying the PTSQ quantization process on our models, we observe expected losses in accuracy around -0.3% for CIFAR-10 and -0.1% for FashionMNIST varied between models and training iterations along with significant increase in robustness. As briefly shown in tables 5.5 and 5.6, there is a significant reduction of the PSNR primarily driven from quantization and further improved by adversarial training. Moreover, quantization not only enhances the adversarial robustness but also dramatically reduces the model sizes as shown in 5.7. With these findings, we prove the benefits of applying sequential adversarial training along with quantization, optimizing the models and increasing their robustness with minimal total accuracy losses of maximum -2%. Finishing this process, we can now prepare our finalized models for deployment on our board assessing its effects.

	CIFAR-10			
	Initial Model	Quantized Version	Adversarial Training 1	Quantized Version
ResNet20	56.68 dB	39.41 dB	53.37 dB	35.46 dB
ResNet56	54.91 dB	38.18 dB	50.32 dB	34.46 dB
MobileNetV2	58.39 dB	41.45 dB	50.55 dB	33.39 dB

Table 5.5: PSNR Comparison between Floating Point and Quantized Versions, CIFAR-10

	FashionMNIST			
	Initial Mode	Quantized Version	Adversarial Training 1	Quantized Version
ResNet20	48.67 dB	38.68 dB	43.76 dB	35.05 dB
ResNet56	47.28 dB	38.05 dB	43.75 dB	34.78 dB
MobileNetV2	45.78 dB	30.53 dB	44.00 dB	29.16 dB

Table 5.6: PSNR Comparison between Floating Point and Quantized Versions, FashionMNIST

	Model Compression	
	Initial Model	Quantized Version
ResNet20	1 Mbytes	355 kbytes
ResNet56	3.1 Mbytes	1.1 Mbytes
MobileNetV2	17.6 Mbytes	4.6 Mbytes

Table 5.7: Model Size Comparison between Floating-Point and Quantized modes

5.5 VitisAI

VitisAI is a software created by Xilinx made for developing and deploying artificial intelligence and machine learning models on Xilinx FPGAs and SoCs. It features multiple tools and libraries to streamline the development process for Xilinx hardware. Its general pipeline can be seen in fig.5.8. Vitis AI features its

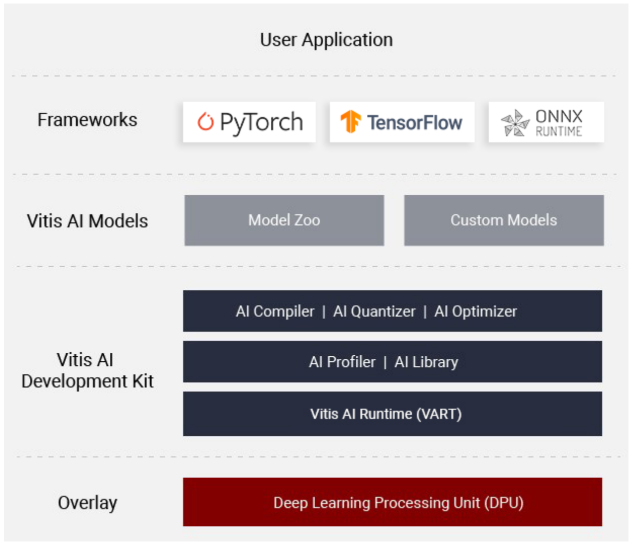


Figure 5.8: Vitis AI Structure, Source: [55]

own library of pre-trained models optimized for use in Xilinx Hardware however, we utilize and modify our own models which is also supported from the suite. Furthermore, it includes an AI Optimizer, Quantizer and Compiler. These are three different tools used for different functions. AI Optimizer fig.5.9 is a tool capable of pruning a model with minimal impact in the accuracy while minimizing model complexity. We do not utilize this tool in order to avoid any unwanted effects in robustness. Additionally, our quantization on models resulted a significant complexity reduction rendering pruning an optional choice.

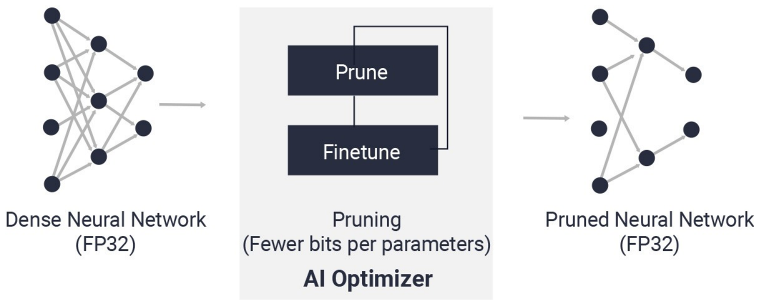
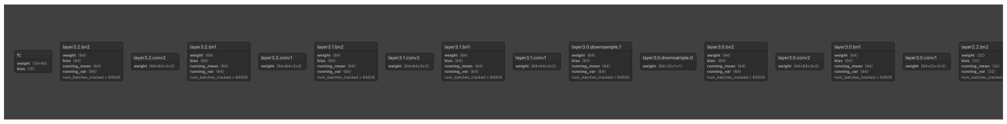
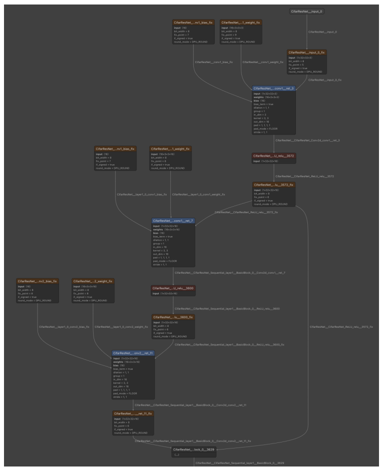

Dense Neural Network (FP32)
Prune
Finetune
Pruning (Fewer bits per parameters)
AI Optimizer
Pruned Neural Network (FP32)

Figure 5.9: Vitis AI Optimizer, Source: [55]

AI Quantizer is a quantization tool required for any model to be deployed on any xilinx hardware. This tool quantizes and evaluates the models using Post Training Static Quantization. It is important to note however, it utilizes a proprietary method of redefining the model's layers and functions with custom-made equivalent ones along with its quantization process. The use of this tool is mandatory in our application and for this reason we utilize the non-quantized 3 times adversarial trained models comparing our quantization process with the AI Quantizer. AI Complier is the tool used for deployment on a targeted board. This tool is used in conjunction with the quantizer where we define our targeted DPU architecture (DPUCVDX8G_ISA3_C32B6 in our case) and the model is compiled accordingly. VitisAI also features an AI Profiler which estimates the resources required to run a specified model on a targeted architecture and whether its possible. Finally, Vitis AI Runtime finalizes the changes required on the model in order to run on our board. Overall utilizing VitisAI allows us to easily convert and deploy networks on hardware devices. Applying the VitisAI Quantization, we get estimated accuracies on how the model performs on our specified hardware. From this estimation we observe close similarities between our quantization and the one applied from the quantizer, expecting these similarities to pass on our device performance. Utilizing Netron, a known application for model inspection, we can see the process applied on ResNet20 on fig.5.10 from its floating point state to the hardware-ready version. We observe a different structure on the quantized version compared to our quantization, however, on closer inspection, the model retains its relative structure with the addition of bias and weight correction nodes as part of the quantization process.


(a) Floating Point Model


(b) Quantized Model

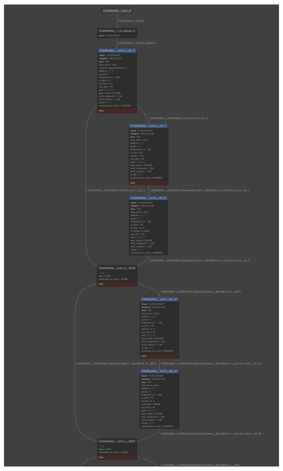

(c) Deployed Model

Figure 5.10: Quantization-Deployment Process

5.6 Versal

Moving to our evaluation board fig.5.11, we must first set it up following the user guide in order to utilize the Vitis-AI mode. Our device supports custom deployment and tuning of the FPGA along with any custom implementations utilizing its resources. It also supports VitisAI libraries however, concurrent use of custom implementations and VitisAI is not possible. Having our board set-up, using C++ and examples provided from Xilinx, we create the source code required to run the models along with the normalization required from CIFAR-10 models.

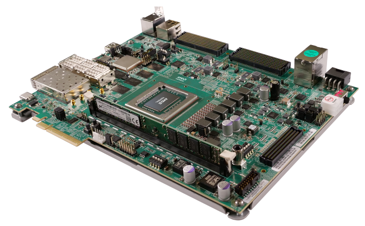


Figure 5.11: Versal VCK190 Board, Source: [56]

Since we run our models on a hardware device we are interested in the accuracy of the models along with the latency and efficiency of our device per model. For this reason, we create another script that allows us to calculate the latency and performance of the device's DPUs. This also allows us to utilize the parallelization support of our device up to a maximum of 6 threads. In contrast to our normal setup where testing on the models is done through a dataset and dataloader, here testing requires the images extracted in a folder and then processed by the model. Due to this limitation, we create multiple scripts for out testing and evaluation purposes. In the following chapter we evaluate our results and achievements.

Chapter 6

Results

In this chapter we provide in-depth metrics and evaluations of our procedures and their effects on the models accuracy and robustness. The following sections are divided per dataset in order to avoid cluttering and mixing of the results since the 3 models are evaluated in 4 steps each. In tables 6.1 and 6.2 we observe the initial accuracies of our pre-trained models in the CIFAR-10 dataset and FashionMNIST along with their starting PSNR values which we use as our base metrics. We aim to preserve the accuracies while reducing the PSNR values to about 30 dB.

	CIFAR-10	
	Top-1 Accuracy	PSNR
ResNet20	92.60 %	56.68 dB
ResNet56	94.37%	54.91 dB
MobileNetV2	94.22%	58.36 dB

Table 6.1: Base Metrics, CIFAR-10

	FashionMNIST	
	Top-1 Accuracy	PSNR
ResNet20	92.24%	48.67 dB
ResNet56	93.72%	47.28 dB
MobileNetV2	93.72%	45.78 dB

Table 6.2: Base Metrics, FashionMNIST

6.1 Adversarial Training

As we detailed in our previous chapter our adversarial training consists of three steps with incremental decreases in our dataset size. In table 6.3 we can see our 3 different sizes the effectiveness of which we evaluate in this section.

	Adversarial Datasets
1st Adversarial Training	30.000 examples
2nd Adversarial Training	20.000 examples
3rd Adversarial Training	10.000 examples

Table 6.3: Base Metrics, CIFAR-10

6.1.1 CIFAR-10

Examining each training iteration one by one for each model on each dataset is not only inefficient but also tiresome. As such, we provide all results combined in a compact form for each model, examining each one in the process. Starting with the training iterations we can observe the accuracy reductions in tables 6.4 to 6.6. As we observe, ResNet20 loses a total of 1.25% for its top-1 accuracy after our full three-step training

	ResNet20 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	92.60 %			
1st Adversarial Training	92.54 %	-0.06 %		
2nd Adversarial Training	92.00 %	-0.60 %	-0.54 %	
3rd Adversarial Training	91.35 %	-1.25 %	-1.19 %	-0.65 %

Table 6.4: ResNet20 Adversarial Training Accuracy, CIFAR-10

process. This alone is regarded as a minimal loss in accuracy and combined with our achieved robustness increase, which we examine later on, it is a negligible and justifiable loss. ResNet56 in comparison, while still minimal, suffers greater losses in accuracy (2%) in relation to ResNet20. It is important to note however, that ResNet56 is multiple times more complex than ResNet20 rendering the increase in accuracy loss an expected behavior.

	ResNet56 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	94.37 %			
1st Adversarial Training	93.90 %	-0.47 %		
2nd Adversarial Training	93.14 %	-1.23 %	-0.76 %	
3rd Adversarial Training	92.33 %	-2.04 %	-1.57 %	-0.81 %

Table 6.5: ResNet56 Adversarial Training Accuracy, CIFAR-10

	MobileNetV2 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	94.22 %			
1st Adversarial Training	94.17 %	-0.05 %		
2nd Adversarial Training	93.58 %	-0.64 %	-0.59 %	
3rd Adversarial Training	92.79 %	-1.43 %	-1.38 %	-0.79 %

Table 6.6: MobileNetV2 Adversarial Training Accuracy, CIFAR-10

Finally, MobileNetV2 while more complex than both ResNet architectures, features a similarly negligible accuracy loss of 1.43% after the whole process. Additionally, by featuring a higher initial accuracy percentage from ResNet20 and a lower percentage decrease than ResNet56, before evaluating its robustness, it is our highest performing model. Moving on to our PSNR evaluations, in tables 6.7 to 6.9 we can observe the incremental PSNR decrease which translates to increase in robustness. Here our percentages are calculated based on our initial PSNR values which occur from the geometric mean of the adversarial test datasets produced for this reason. ResNet20 exhibits a total robustness increase of 12.1% with only 1.25% trade-

	ResNet20 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	56.68 dB			
1st Adversarial Training	53.37 dB	-5.83 %		
2nd Adversarial Training	50.21 dB	-11.41 %	-5.92 %	
3rd Adversarial Training	49.84 dB**	-12.10 %	-6.65 %	-0.77 %

Table 6.7: ResNet20 Adversarial Training PSNR Evaluation, CIFAR-10

off in accuracy. Our objective is to achieve higher robustness percentages however, this relation between robustness and accuracy is adequate enough to continue. Also note the PSNR value of our 3rd Training is altered for reasons we explain after evaluating the other models. In ResNet56 we observe similar percentages

	ResNet56 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	54.91 dB			
1st Adversarial Training	50.37 dB	-8.26 %		
2nd Adversarial Training	48.50 dB	-11.67 %	-3.71 %	
3rd Adversarial Training	46.83 dB	-14.71 %	-7.02 %	-3.44 %

Table 6.8: ResNet56 Adversarial Training PSNR Evaluation, CIFAR-10

with 14% increase in robustness while 2% decrease in accuracy. It is important to note however, ResNet20 results in 49.82 dB for our final adversarial images while ResNet56 results in 46.83 dB. This difference while relatively small, shows that ResNet56 yields better results in respect to ResNet20 meaning increased robustness.

	MobileNetV2 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	54.45 dB			
1st Adversarial Training	50.55 dB	-7.16 %		
2nd Adversarial Training	48.29 dB	-11.3 %	-4.47 %	
3rd Adversarial Training	46.79 dB	-14.07 %	-7.43 %	-3.1 %

Table 6.9: MobileNetV2 Adversarial Training PSNR Evaluation, CIFAR-10

Finally, on MobileNetV2 we observe a similar improvement at 14% with minimal losses in accuracy. Accounting for the complexity and accuracy of MobileNetV2, it is the best performing model of the three. However, it must be noted that these percentages are in relation to the starting values of each model meaning that while our final PSNR values are lower, the distribution of these values is also important. Utilizing the geometrical mean for our PSNR evaluations, we produce an accurate representation of the data however, due to its ability to remain unaffected over sharp fluctuations we must also evaluate the distribution of PSNR values on our adversarial test sets in figs. 6.1 to 6.3.

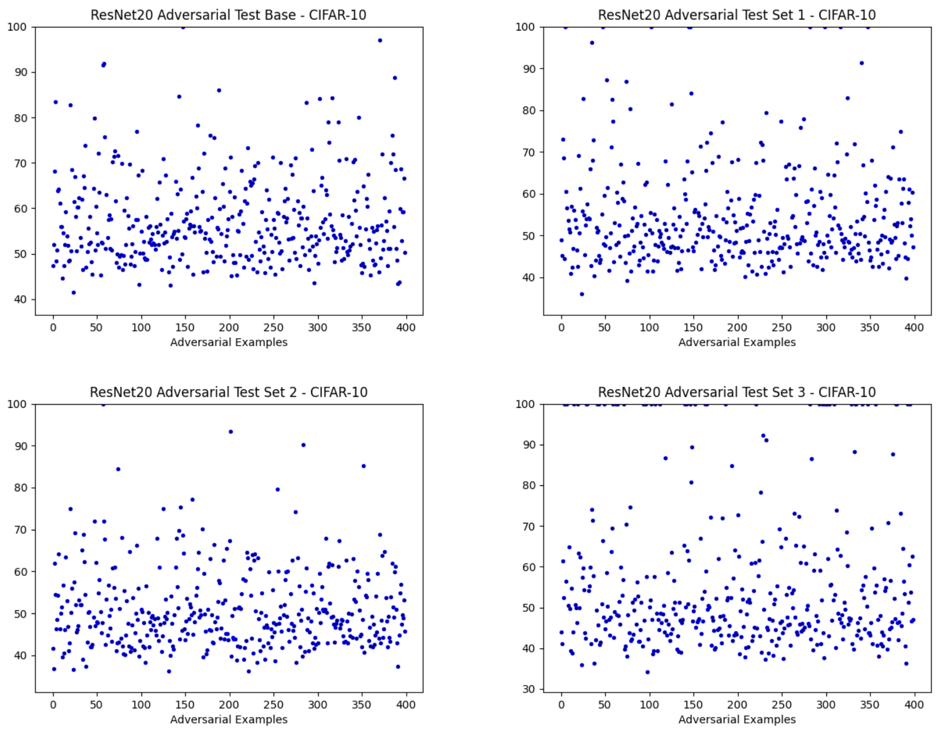


Figure 6.1: ResNet20 PSNR Evaluation - CIFAR-10

Observing the distributions of PSNR in the figures above, we can see multiple entries at 100dB for our third dataset (d). An 100dB value means the image is unchanged from its original form. This is caused due to the decreased accuracy of our network misclassifying these images without requiring any noise.

Taking them into account, our resulting PSNR would be 53.99 dB however, our robustness evaluation accounts only for adversarial images thus we exclude network misclassifications since they happen regardless of any robustness enhancing technique. Aside this change, as we observe from the figures, with each adversarial training iteration, values are not only decreased, but also they are incrementally closely arranged towards the lower values.

Evaluating the distributions of ResNet56 we observe an interesting result. While featuring similar metrics to ResNet20, it has minimal misclassifications and very close arrangements of the images toward lower values. These attributes follow between training iterations with each step lowering the PSNR values on all examples. Additionally, we observe an even lower number of images with high remaining values, highlighting the capabilities of ResNet56 for complex image classification.

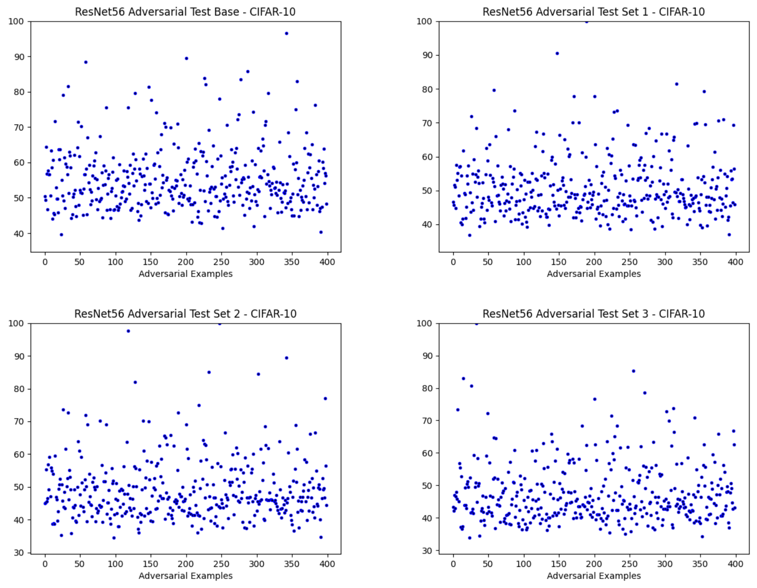


Figure 6.2: ResNet56 PSNR Evaluation - CIFAR-10

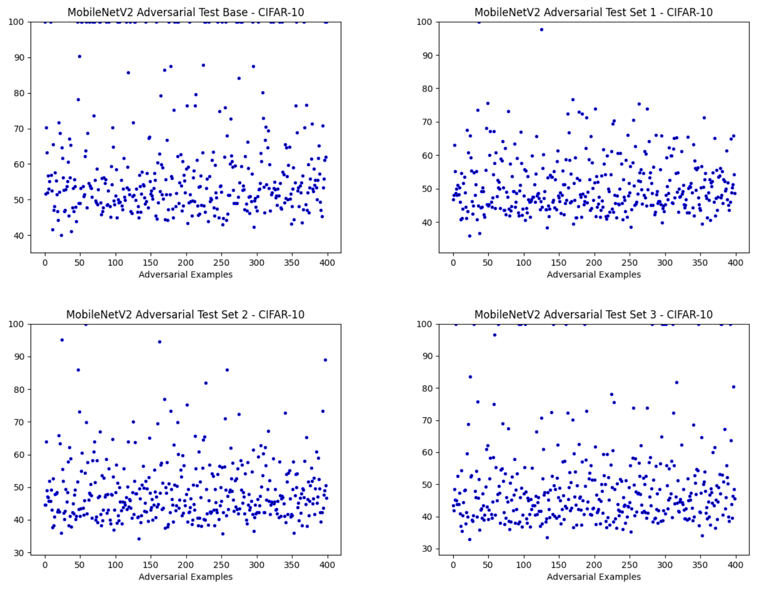


Figure 6.3: MobileNetV2 PSNR Evaluation - CIFAR-10

Finally, MobileNetV2 features even tighter groupings of the images with lower values across iterations, signifying the importance of complex but lightweight models. Moreover, when compared with ResNet56 we observe similar PSNR results with even less images retaining their higher values rendering MobileNetV2 a slightly more robust model.

6.1.2 FashionMNIST

The FashionMNIST dataset has monochrome less complex images. For this reason we expect lower losses in accuracy and lower PSNR values compared to CIFAR-10. In tables 6.10 to 6.12, we observe the accuracy differences between training iterations on each model. ResNet20 features an unexpected increase in accuracy

	ResNet20 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	92.54 %			
1st Adversarial Training	93.52 %	+0.99 %		
2nd Adversarial Training	93.35 %	+0.81 %	-0.17 %	
3rd Adversarial Training	93.01 %	+0.47 %	-0.51 %	-0.34 %

Table 6.10: ResNet20 Adversarial Training Accuracy, FashionMNIST

from our first training iteration. As we detailed in the previous chapter, we trained the models on Fashion-MNIST with adequate enough accuracies. This increase in accuracy simply signifies the ability of the model to reach higher percentages with further optimization of the hyperparameters. Furthermore, we can also see the decrease in accuracy is exactly 0.5% which is easily regarded as totally negligible. This preservation of accuracy carries over to the other models highlighting the simplicity of FashionMNIST over CIFAR-10 and the expected increase in robustness.

	ResNet56 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	93.72 %			
1st Adversarial Training	93.64 %	-0.08 %		
2nd Adversarial Training	93.63 %	-0.09 %	-0.01 %	
3rd Adversarial Training	93.29 %	-0.43 %	-0.35 %	-0.34 %

Table 6.11: ResNet56 Adversarial Training Accuracy, FashionMNIST

Similarly, ResNet56 has only 0.45% accuracy loss whereas in this case we do not observe any accuracy increases.

	MobileNetV2 Accuracy Evaluation			
	Top-1 Accuracy	Delta in relation to above percentages		
Base Accuracy	93.72 %			
1st Adversarial Training	93.52 %	-0.20 %		
2nd Adversarial Training	93.40 %	-0.32 %	-0.12 %	
3rd Adversarial Training	93.31 %	-0.41 %	-0.21 %	-0.09 %

Table 6.12: MobileNetV2 Adversarial Training Accuracy, FashionMNIST

Finally, MobileNetV2 also has minute accuracy losses across training iterations. Moving on our PSNR evaluations, tables 6.13 to 6.15 provide the corresponding details.
As expected, we observe lower PSNR values signifying both increased robustness and dataset simplicity compared to CIFAR-10. Additionally, the model's robustness increased significantly up to 22% with low enough PSNR values where human detection starts to be possible.

	ResNet20 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	48.67 dB			
1st Adversarial Training	43.76 dB	-10.08 %		
2nd Adversarial Training	40.51 dB	-16.76 %	-7.42 %	
3rd Adversarial Training	37.76 dB	-22.41 %	-13.71 %	-6.78 %

Table 6.13: ResNet20 Adversarial Training PSNR Evaluation, FashionMNIST

ResNet56 also has a significant robustness increase of relatively the same type with only 0.4 dB difference in comparison to ResNet20.
Finally, MobileNetV2 in contrast to the CIFAR-10, exhibits the least improvement in terms of robustness at 14% with about 2dB PSNR difference compared to other models.
Assessing the PSNR distributions for FashionMNIST, we observe the results on figs. 6.4 to 6.6.
Compared to the initial distribution of ResNet20, we achieve both value reduction and increased robustness on complex images with most images not exceeding 50dB on the 3rd iteration.

	ResNet56 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	47.28 dB			
1st Adversarial Training	43.75 dB	-7.46 %		
2nd Adversarial Training	40.34 dB	-14.67 %	-7.79 %	
3rd Adversarial Training	37.36 dB	-20.98 %	-14.06 %	-7.38 %

Table 6.14: ResNet56 Adversarial Training PSNR Evaluation, FashionMNIST

	MobileNetV2 PSNR Evaluation			
	PSNR	Delta in relation to above percentages		
Base	45.78 dB			
1st Adversarial Training	44.00 dB	-3.88 %		
2nd Adversarial Training	41.52 dB	-9.30 %	-5.63 %	
3rd Adversarial Training	38.98 dB	-14.85 %	-11.40 %	-6.11 %

Table 6.15: MobileNetV2 Adversarial Training PSNR Evaluation, FashionMNIST

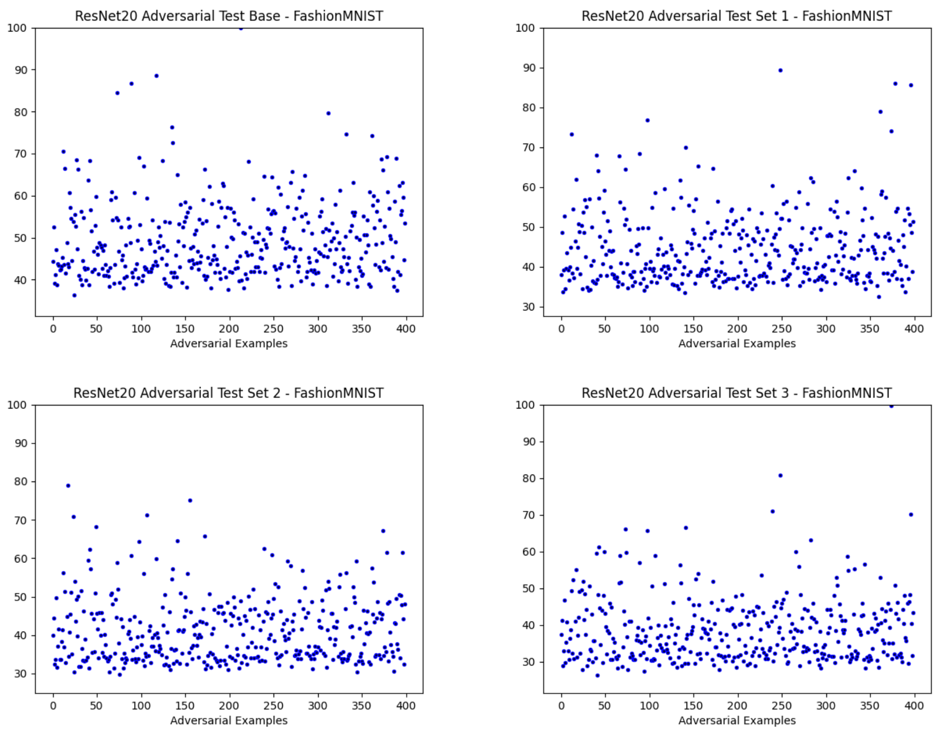


Figure 6.4: ResNet20 PSNR Evaluation - FashionMNIST

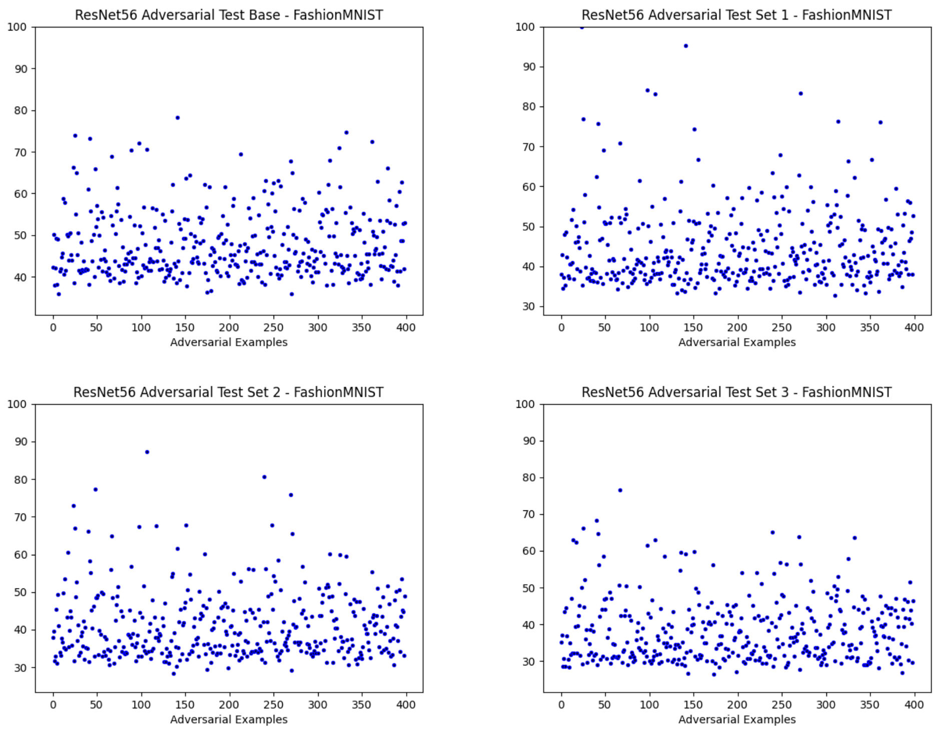


Figure 6.5: ResNet56 PSNR Evaluation - FashionMNIST

ResNet56 while featuring similar PSNR mean values, has a maximum of ≈40dB for most of the images, providing improved robustness further highlighting the importance of complex deep neural networks.

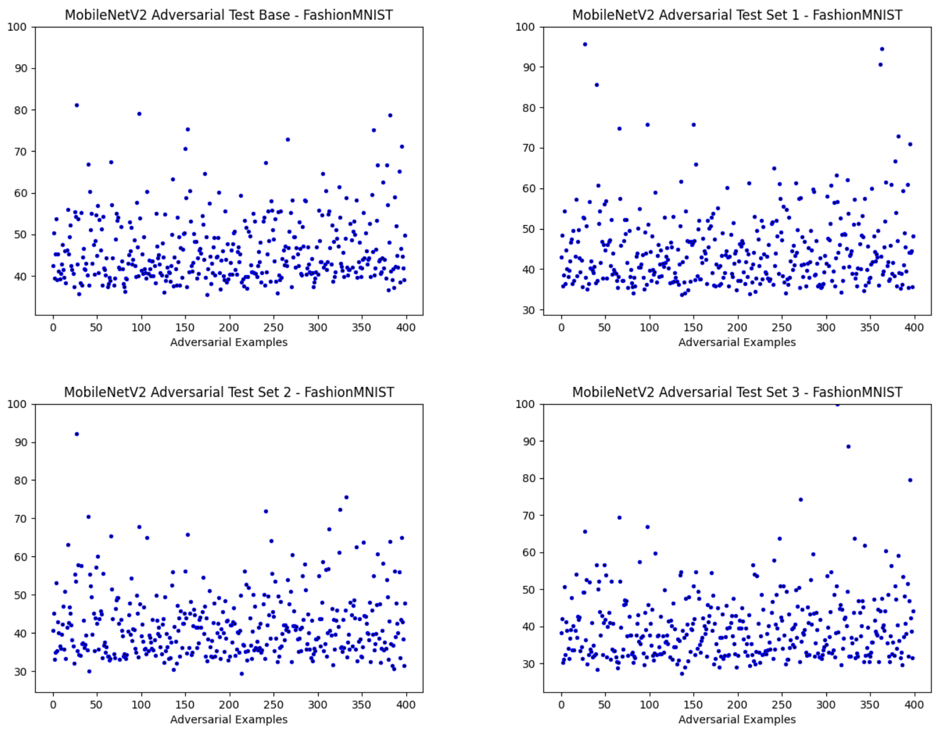


Figure 6.6: MobileNetV2 PSNR Evaluation - FashionMNIST

Lasty, MobileNetV2 exhibits an interesting behavior with FashionMNIST. While its robustness has increased, it presents relatively higher values compared to the ResNet models. The difference however lies in the value distribution where MobileNetV2 has significantly less dispersion compared to the others meaning it is equally capable of handling both simple and complex images on the dataset.

6.2 Quantization

Building on the robustness achieved through adversarial training, we apply quantization on each step of our models evaluating its effects.

6.2.1 CIFAR-10

Applying Quantization on our models we observe the expected accuracy losses as shown in table 6.16.

While accuracy losses are relatively low, we observe minimal impact on ResNet20 in comparison to the other two models while ResNet56 has the largest impact. Excluded from our table for clarity, the overall losses from our starting models to the final trained and quantized ones can be seen in table 6.17.

This happens due to the differences in parameters and complexity between models with ResNet20 being

	Quantization Accuracy Loss			
	Standard	Quantized Step 1	Quantized Step 2	Quantized Step 3
ResNet20	92.33 % -0.27%	92.17 % -0.37%	91.87 % -0.13%	91.28 % -0.07%
ResNet56	94.22 % -0.15%	93.64 % -0.25%	92.81 % -0.33%	92.24 % -0.09%
MobileNetV2	94.03 % -0.19%	93.61 % -0.56%	93.28 % -0.30%	92.37 % -0.42%

Table 6.16: Quantized Models Accuracy Evaluation, CIFAR-10

	Finallized Losses	
	Initial Model	Quantized Retrained Version
ResNet20	92.60 %	91.28 % -1.32%
ResNet56	94.37 %	92.24 % -2.13%
MobileNetV2	94.22 %	92.37 % -1.85%

Table 6.17: Accuracy Losses from Starting to Final models, CIFAR-10

marginally smaller and ResNet56 more complex. However, such accuracy losses are expected, we are more interested in effects on the robustness of the models. Generating new adversarial examples on the quantized models, we calculate the gmean PSNR in table 6.18

	Quantized PSNR Evaluation			
	Standard	Quantized Step 1	Quantized Step 2	Quantized Step 3
ResNet20	39.41 dB -30.00%	35.46 dB -33.55%	32.85 dB -34.57%	32.64 dB -34.51%
ResNet56	38.18 dB -30.04%	34.46 dB -31.58%	32.58 dB -32.82%	31.31 dB -33.14%
MobileNetV2	41.45 dB -23.87%	33.39 dB -33.94%	31.43 dB -34.91%	32.18 dB -31.22%

Table 6.18: Quantized Models PSNR Evaluation, CIFAR-10

Observing our results we can see dramatic differences in the PSNR compared to the floating-point counterparts. While models produced varying results in their floating-point state, their quantized versions offer similar robustness enhancements with ResNet20 as the weakest and ResNet56 as the most robust. Calculating the overall difference in PSNR from our initial floating point models to our quantized ones, we achieve **+42.41% adversarial robustness for ResNet20, +42.97% for ResNet56 and +44.88% for MobileNetV2**. However we still need to evaluate the PSNR distributions as shown in figs. 6.7 to 6.9

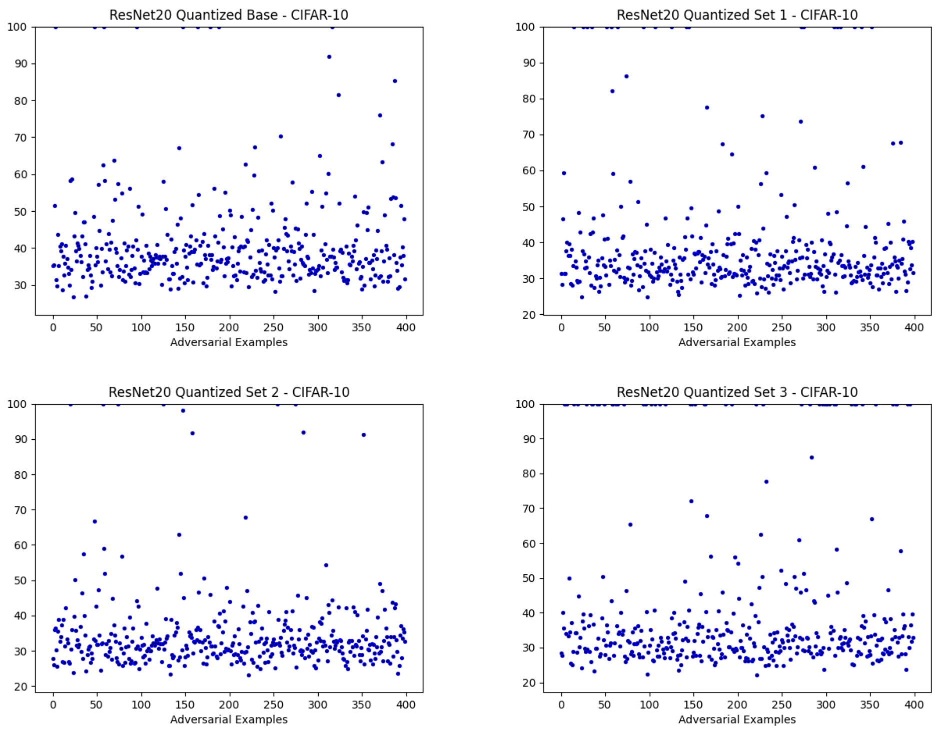


Figure 6.7: ResNet20 PSNR Evaluation - CIFAR-10

As we can see ResNet20 in contrast to its original version presents very low PSNR values with minimal distribution directly translating to greatly increased adversarial robustness. However, as we can see in its final quantized version, there are multiple examples of 100dB which as we explained before are misclassifications on the network due to its accuracy. We don't exclude these examples in the quantized version as quantization along with the robustness, directly affects the accuracy of a model.

Moving to ResNet56, preserving its capabilities on complex images from the floating-point version, we observe similar increases in robustness with barely any misclassifications.

Finally, MobileNetV2 features multiple misclassifications affecting its accuracy while also similarly to ResNet56 it features exceptional robustness performance. Evaluating these models we conclude ResNet56 offers better performance on CIFAR-10 while all models exhibit exceptional adversarial robustness through our process. In fig. 6.10 we include an accuracy to robustness evaluation for all models for each training iteration and quantization in order to more intuitively understand their effects.

We also observe that this robustness leads to noisy enough images for human detection. To understand this more intuitively lets observe some examples from the datasets in figs. 6.11 to 6.13 and some ineffective ones in figs. 6.14 and 6.15.

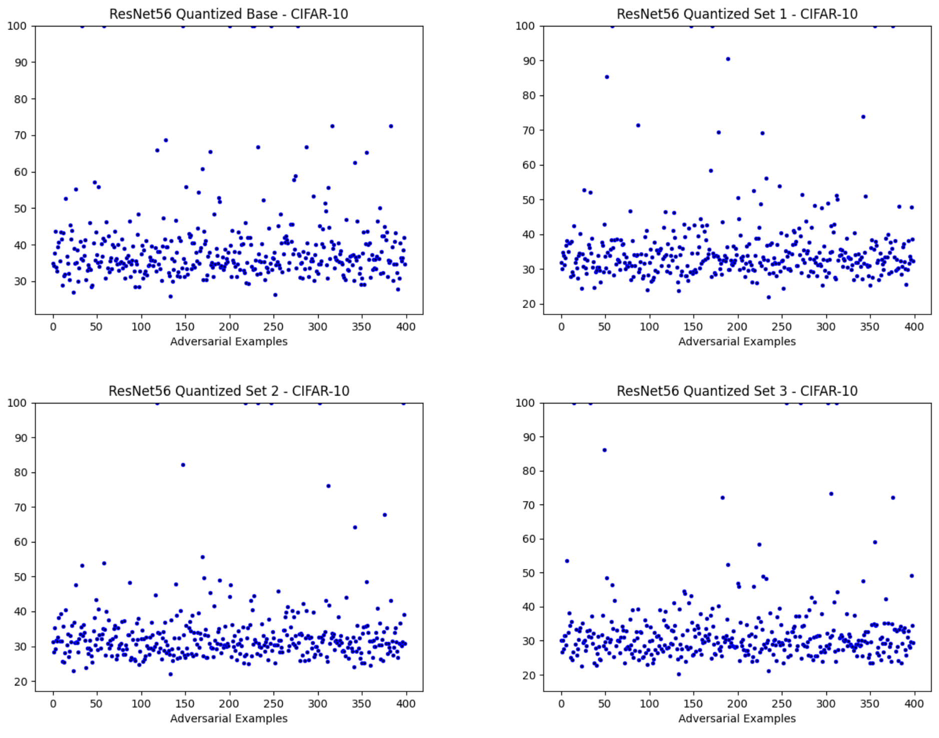


Figure 6.8: ResNet56 PSNR Evaluation - CIFAR-10

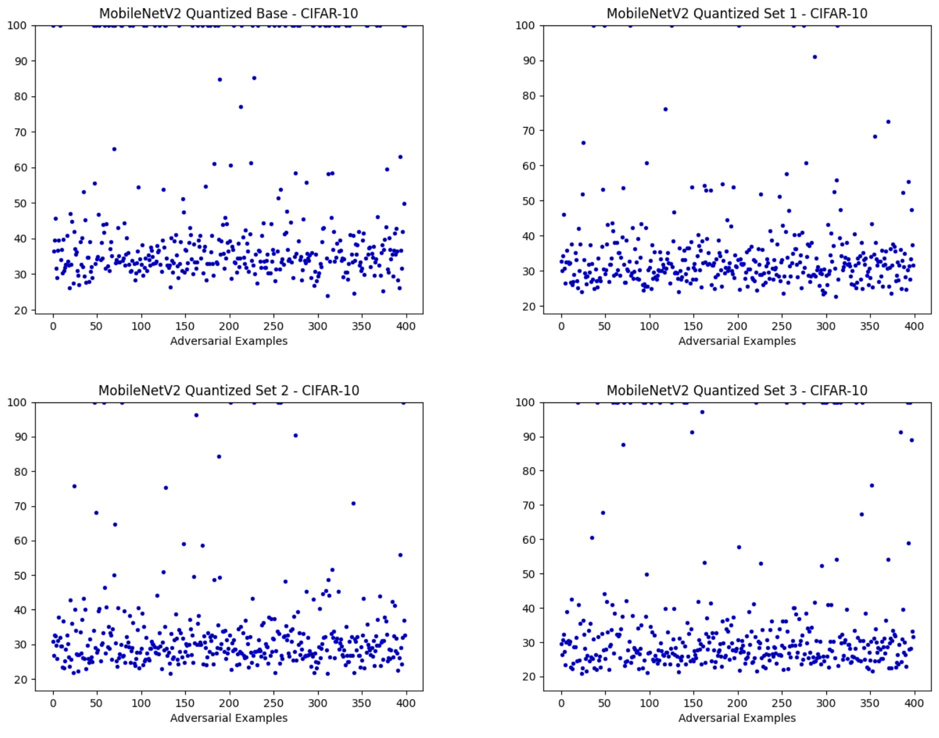


Figure 6.9: MobileNetV2 PSNR Evaluation - CIFAR-10

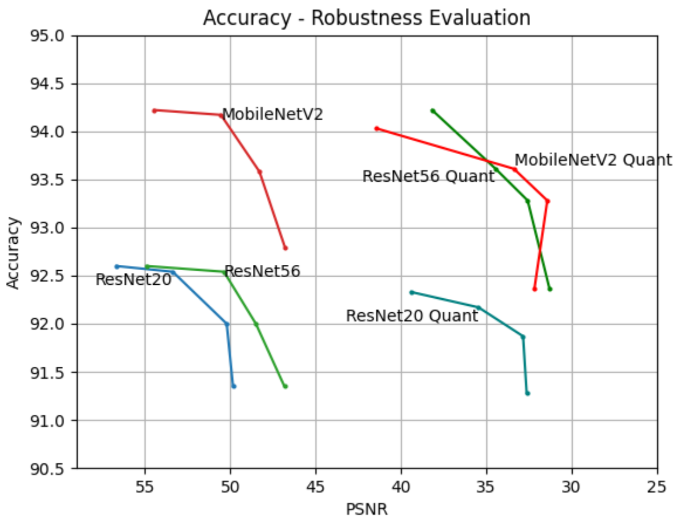
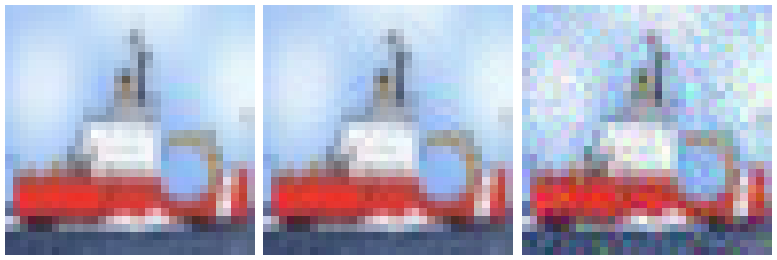
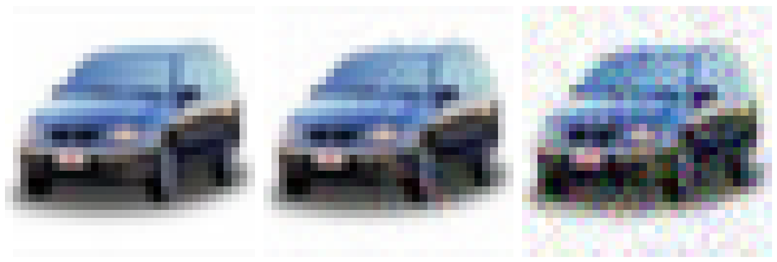


Figure 6.10: Accuracy - Robustness Evaluation - CIFAR-10



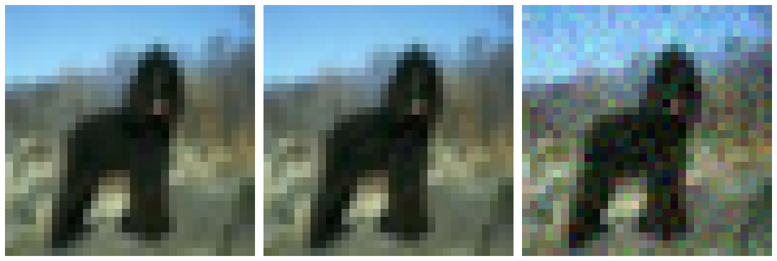
(a) Original Image (b) Second eval set (c) Final Image

Figure 6.11: ResNet20 Robust Example - CIFAR-10



(a) Original Image (b) Second eval set (c) Final Image

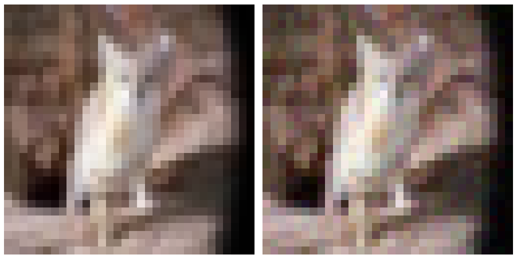
Figure 6.12: ResNet56 Robust Example - CIFAR-10



(a) Original Image (b) Second eval set (c) Final Image

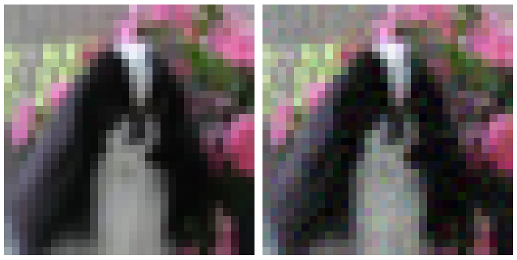
Figure 6.13: MobileNetV2 Robust Example - CIFAR-10

While initially the differences are imperceptible, we can clearly see the noise added from HopSkipJumpAttack in our finalized 3-step adversarial trained quantized models.



(a) Original Image (b) Final Image

Figure 6.14: Undetectable Example - CIFAR-10



(a) Original Image (b) Final Image

Figure 6.15: Undetectable Example - CIFAR-10

6.2.2 FashionMNIST

Applying the same evaluations in FashionMNIST, table 6.19 presents the accuracy differences while table 6.21 includes the PSNR differences.

	Quantization Accuracy Loss			
	Standard	Quantized Step 1	Quantized Step 2	Quantized Step 3
ResNet20	92.21 % -0.03%	93.44 % -0.08%	93.27 % -0.08%	92.97 % -0.04%
ResNet56	93.74 % +0.02%	93.60 % -0.04%	93.45 % -0.18%	93.12 % -0.17%
MobileNetV2	93.59 % -0.13%	93.28 % -0.24%	93.11 % -0.29%	93.15 % -0.16%

Table 6.19: Quantized Models Accuracy Evaluation, FashionMNIST

Compared to CIFAR-10 here we observe minimal accuracy losses mostly due to the fact FashionMNIST has only one-channel, monochrome images. The finalized accuracy changes from the initial floating-point models to the quantized retrained ones, are available in 6.20

	Finallized Losses	
	Initial Model	Quantized Retrained Version
ResNet20	92.24 %	92.97 % +0.73%
ResNet56	93.72 %	93.12 % -0.60%
MobileNetV2	93.72 %	93.15 % -0.57%

Table 6.20: Accuracy Losses from Starting to Final models, FashionMNIST

	Quantized PSNR Evaluation			
	Standard	Quantized Step 1	Quantized Step 2	Quantized Step 3
ResNet20	38.68 dB -20.50%	35.05 dB -19.90%	32.20 dB -20.51%	29.65 dB -21.47%
ResNet56	38.05 dB -19.52%	34.78 dB -20.50%	32.20 dB -20.17%	29.43 dB -21.22%
MobileNetV2	30.53 dB -33.33%	29.16 dB -33.72%	27.32 dB -34.20%	24.89 dB -36.14%

Table 6.21: Quantized Models PSNR Evaluation, FashionMNIST

Here we observe another interesting behavior. While our percentages in PSNR are smaller compared to CIFAR-10, which would imply a smaller effect of quantization on the models, due to initially smaller PSNR values, we achieve an even greater adversarial robustness enhancement. In figs. 6.16 to 6.18 we observe the distributions of PSNR for our models expecting similarly improved results as with CIFAR-10.

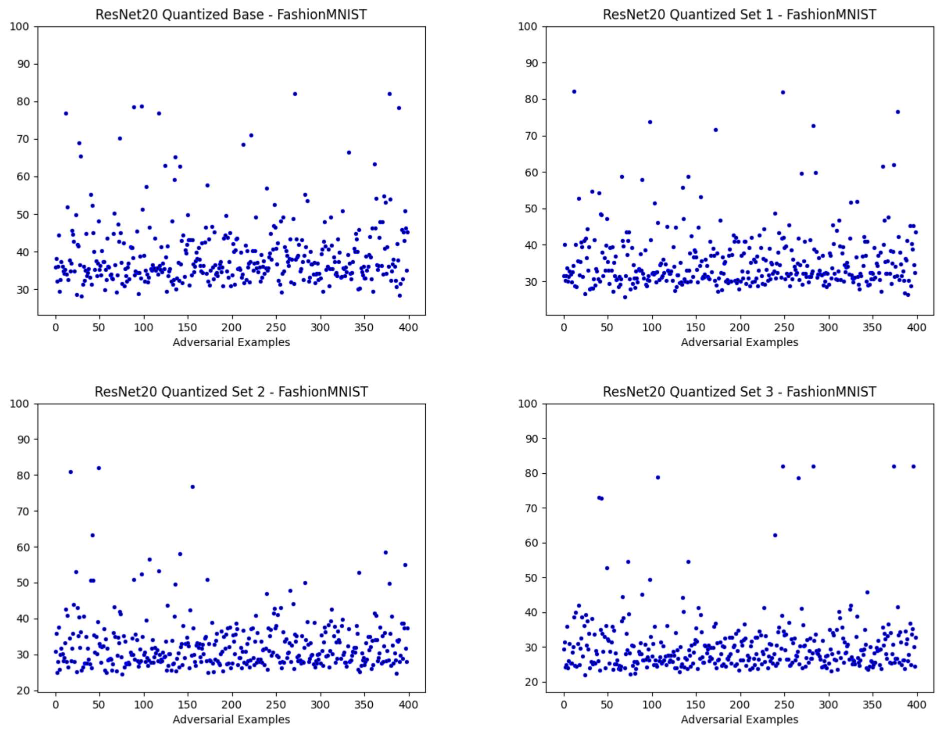


Figure 6.16: ResNet20 PSNR Evaluation - FashionMNIST

Evaluating our graphs for ResNet20 we can see the cluttering of entries toward lower values along with a small 4% of images remaining over 35 dB. This further proves the effectiveness of our methods of enhancing adversarial robustness.

Moving on ResNet56, here the values are not only lower and more cluttered but there is only 3% of images regarded as imperceptible, leveraging the complexity of the network further improving its adversarial robustness.

Finally, MobileNetV2 featuring even lower PSNR values has the same 3% of imperceptible images.
With geometric mean values of under 30 dB for FashionMNIST, we achieve our proposed robustness level of human detection of adversarial examples. In fig. 6.19 we observe the effects of retraining and robustness on our networks while in order to showcase their effect, figs. 6.20 to 6.22 feature some example images from each model on different stages and fig. 6.23 features an image that is still imperceptible.

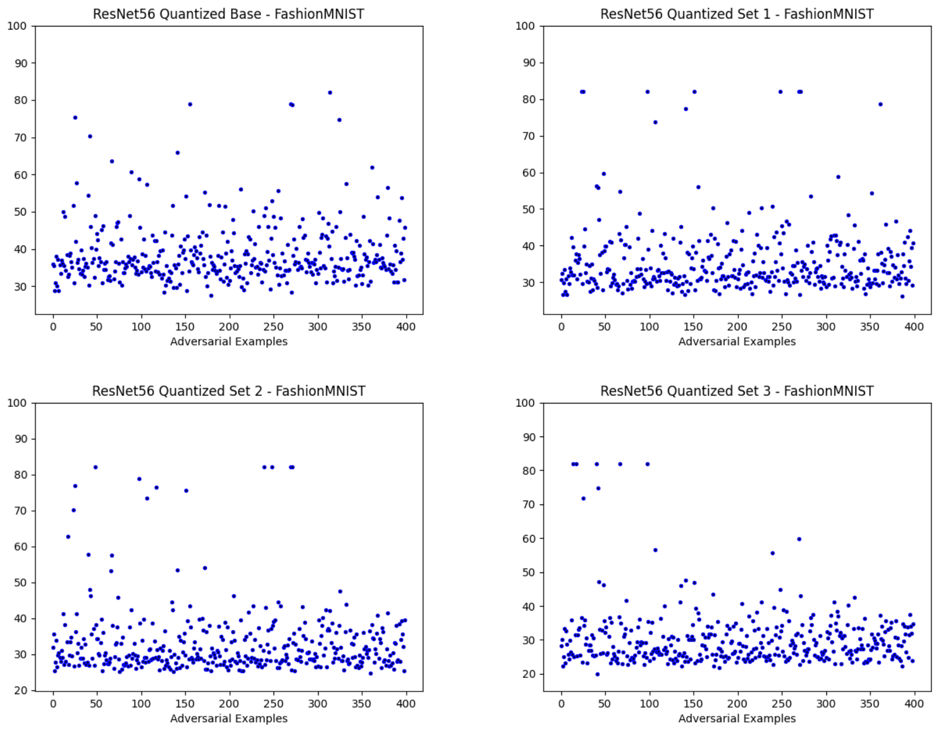


Figure 6.17: ResNet56 PSNR Evaluation - FashionMNIST

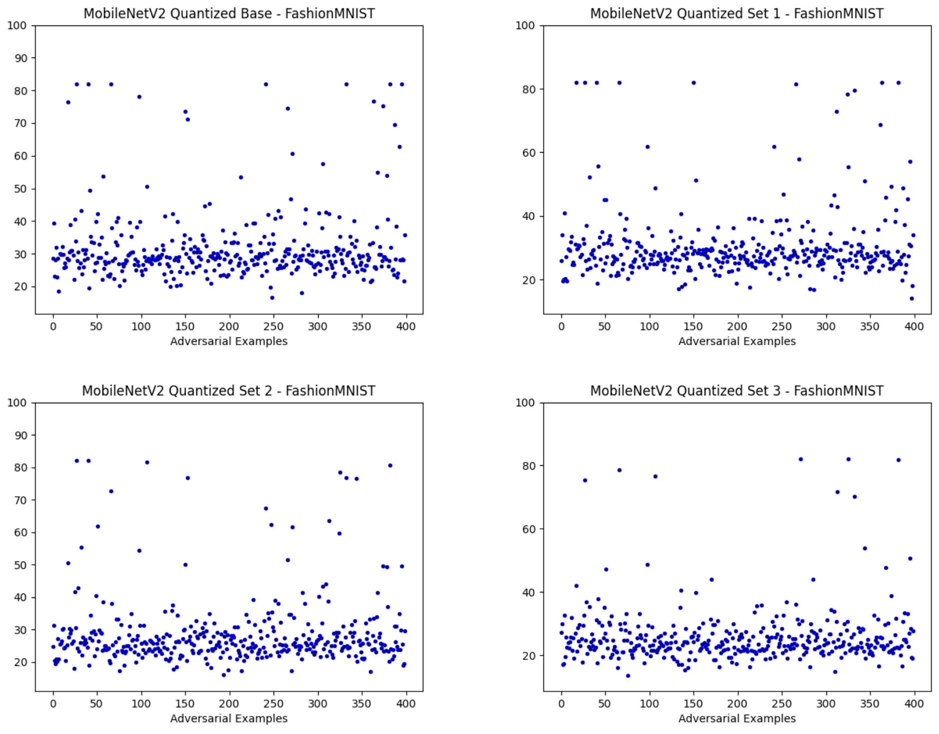


Figure 6.18: MobileNetV2 PSNR Evaluation - FashionMNIST

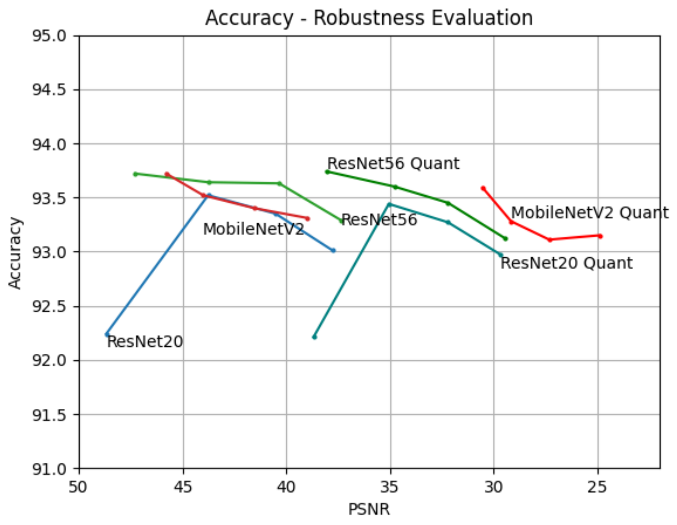
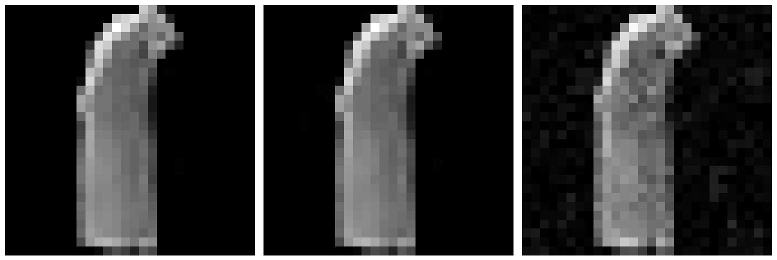
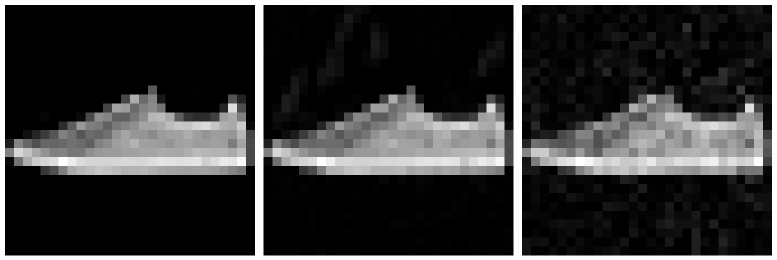


Figure 6.19: Accuracy - Robustness Evaluation - FashionMNIST



(a) Original Image (b) Second eval set (c) Final Image

Figure 6.20: ResNet20 Robust Example - FashionMNIST



(a) Original Image (b) Second eval set (c) Final Image

Figure 6.21: ResNet56 Robust Example - FashionMNIST

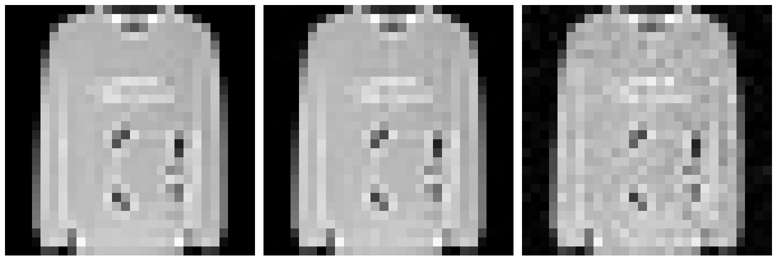

(a) Original Image (b) Second eval set (c) Final Image

Figure 6.22: MobileNetV2 Robust Example - FashionMNIST

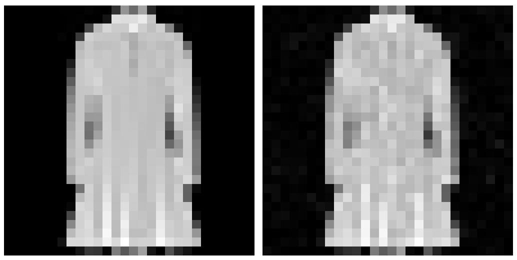

(a) Original Image (b) Final Image

Figure 6.23: Undetectable Example - FashionMNIST

6.3 Versal

Having our method evaluated on all three models in two different datasets we can conclude on its effectiveness as an adversarial defense technique. Showing promising results on our computers, we now assess the performance of the models on the Versal Platform. However, we only need our finalized models omitting any intermediate steps. It is also important to note that we can't generate adversarial examples on our platform in order to evaluate its robustness performance. For this reason we resort to evaluating the performance on the adversarial test datasets used for our quantized models. Comparing the expected to be 0 accuracies of these datasets, we can estimate if the robustness has increased or decreased compared to our quantized variant. We also omit all VitisAI estimations on accuracy due to significant discrepancies.

6.3.1 CIFAR-10

Starting on CIFAR-10 models, we can see the accuracy and performance of the networks in tables 6.22 and 6.23. Note that latency and FPS (frames per second) are closely related, with latency referring to the time required to process one image and FPS to the images that can be processed in one second.

		CIFAR-10 Accuracy					
		Floating-Point		Quantized Retrained		**Versal**	
		Top - 1	Top - 5	Top - 1	Top - 5	Top - 1	Top - 5
ResNet20	Accuracy	91.35 %	99.56 %	91.28 %	99.58 %	91.33 %	99.55 %
ResNet56	Accuracy	92.33 %	99.67 %	92.24 %	99.62 %	92.13 %	99.57 %
MobileNetV2	Accuracy	92.79 %	99.77 %	92.37 %	99.77 %	92.86 %	99.73 %

Table 6.22: Versal Deployed Models Accuracy Evaluation, CIFAR-10

As we can see, the complexity of each model directly affects the performance of the device. With ResNet20 being a lightweight simple model we observe the significant latency differences compared to the more complex ResNet56 and MobileNetV2 models. Furthermore, we also note that while ResNet20 is accelerated by the use of multiple threads showing up to 250% performance increase, the other two models are unaffected with the use of more than 2 threads. This is mostly attributed to the complexity and design differences between models. Assessing the accuracy on our device however, we can easily see minimally affected to better results when compared with our starting floating point models. While being deployed on a hardware platform is a huge factor, our models exhibit almost the same accuracies as their pre-trained counter parts with ResNet56 losing 0.2% and MobileNetV2 gaining 0.07%. Assuming the robustness attributes have passed on these

		CIFAR-10 Performance					
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
ResNet20	Latency	43.36 us	25.31 us	19.71 us	18.55 us	17.54 us	20.89 us
	Average FPS	23 062	39 504	50 722	53 893	57 010	47 852
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
ResNet56	Latency	61.33 us	35.47 us	35.17 us	35.31 us	35.23 us	35.28 us
	Average FPS	16 303	28 188	28 429	28 320	28 381	28 342
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
MobileNetV2	Latency	107.13 us	81.13 us	81.10 us	81.05 us	80.98 us	81.20 us
	Average FPS	9 334	12 325	12 330	12 338	12 348	12 315

Table 6.23: Versal Deployed Models Performance Evaluation, CIFAR-10

models, we have a effectively enhanced the adversarial robustness of our networks with minimal-to-none losses and simplified them for use in resource-limited devices. In order to verify the state of robustness we turn to tables 6.24 and 6.25 comparing the accuracy (top-1, top-5) of our robust quantized networks on the adversarial sets (both test and train) with the versal ones.

		CIFAR-10			
		test 0	test 1	test 2	test 3
ResNet20	Quantized Model	90.75, 99.75	90, 99.75	87, 99.75	71.75, 99.75
	Versal Model	90.22, 99.75	89.22, 99.25	89.71, 99.75	73.43, 99.75
ResNet56	Quantized Model	93.25, 99.75	92.25, 99.75	90.75, 99.75	75.25, 99.75
	Versal Model	92.23, 100	90.98, 99.75	89.47, 99.75	72.9, 100
MobileNetV2	Quantized Model	92.25, 100	92.5, 100	89.75, 99.5	76.5, 99.25
	Versal Model	92.98, 100	91.72, 100	91.47, 99.74	76.19, 99.24

Table 6.24: PSNR Evaluation adversarial test accuracy

Evaluating both tables, we can see the accuracies closely match for ResNet20 and MobileNetV2 while for ResNet56, Versal is approximately 1% lower on both test and train set. However we also notice an interest result. Since these datasets are generated on the corresponding models, while high accuracies on previous datasets for the trained model signifies its increased robustness, its accuracy on the adversarial dataset based on it should be close 0. In our case, none of our quantized models have this low accuracies with the lowest being around 75% for the test set. This happens due to the normalization step before the inference. Normalization is a required step for the CIFAR-10 dataset and due to restrictions on how our data are loaded on both versal and our quantized models, it is applied right before the inference. Meaning when the adversarial examples are evaluated, they are normalized with the CIFAR-10 predefined mean and std

		CIFAR-10			
		train 0	train 1	train 2	train 3
ResNet20	Quantized Model	100, 100	99.75, 100	99.5, 100	85, 100
	Versal Model	100,100	99.49, 100	98.74, 100	83.70, 100
ResNet56	Quantized Model	100, 100	100, 100	99.25, 100	80.5, 100
	Versal Model	99.75, 100	99.5, 100	97.24, 100	79.20, 100
MobileNetV2	Quantized Model	100, 100	100, 100	99.25, 100	46,75.52
	Versal Model	100, 100	100, 100	99.74, 100	45.61, 75.68

Table 6.25: PSNR Evaluation adversarial train accuracy

resulting in the observed increased accuracy. Additionally, since normalization is a preprocessing step, we also find that utilizing preprocessing along with our procedures further enhances the robustness of our models.

Having said that, with accuracies this related between our quantized dataset and versal we can safely assume the robustness enhancements have passed on the Versal variants.

6.3.2 FashionMNIST

Moving on the FashionMNIST dataset where no Normalization is present we observe our accuracy differences in tables 6.26 and 6.27.

		FashionMNIST Accuracy					
		Floating-Point		Quantized Retrained		**Versal**	
		Top - 1	Top - 5	Top - 1	Top - 5	Top - 1	Top - 5
ResNet20	Accuracy	93.01 %	99.68 %	92.97 %	99.66 %	92.98 %	99.66 %
ResNet56	Accuracy	93.29 %	99.75 %	93.12 %	99.76 %	93.21 %	99.65 %
MobileNetV2	Accuracy	93.31 %	99.44 %	93.15 %	99.45 %	93.08 %	99.39 %

Table 6.26: Versal Deployed Models Accuracy Evaluation, FashionMNIST

		FashionMNIST Performance					
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
ResNet20	Latency	40.37 us	23.09 us	20.26 us	18.54 us	19.44 us	21.50 us
	Average FPS	24 770	43 293	49 351	53 936	51 425	46 491
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
ResNet56	Latency	55.89 us	31.37 us	31.34 us	31.20 us	31.27 us	31.18 us
	Average FPS	17 891	31 869	31 900	32 050	31 979	32 065
		1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
MobileNetV2	Latency	101.94 us	77.12 us	77.12 us	77.03 us	77.02 us	77.03 us
	Average FPS	9 809	12 965	12 966	12 982	12 982	12 981

Table 6.27: Versal Deployed Models Performance Evaluation, FashionMNIST

Similarly to CIFAR-10, we observe minimal changes in accuracy in comparison to our pre-trained models with this time a maximum loss of 0.23% on MobileNetV2. Regarding the performance, we observe slightly improved latency and FPS from CIFAR however this mostly due to the different dataset while still threading doesn't yield better results for ResNet56 and MobileNetV2.

In tables 6.28 and 6.29 we can assess the robustness relation to our quantized models where this time there is no normalization and we expect our accuracies to be near 0.

We observe the expected behavior of our quantized models of near 0 accuracy on the corresponding adversarial sets. However, we also observe the deployed models producing high accuracies on these datasets with zero

		FashionMNIST			
		test 0	test 1	test 2	test 3
ResNet20	Quantized Model	92.5, 99.75	93, 99.75	92, 99.75	5.5, 99.75
	Versal Model	92.73, 99.75	92.48, 99.75	92.48, 99.75	74.18, 99.75
ResNet56	Quantized Model	94.75, 99.75	94.25, 99.75	93.75, 99.5	4, 100
	Versal Model	94.48, 99.75	93.98, 99.75	93.48, 99.5	85.21, 100
MobileNetV2	Quantized Model	93, 99.25	92.75, 99.75	93.75, 99.5	3.75, 99.75
	Versal Model	92.73, 99.25	93.23, 99.5	92.98, 99.5	87.72, 99.8

Table 6.28: PSNR Evaluation adversarial test accuracy

		FashionMNIST			
		train 0	train 1	train 2	train 3
ResNet20	Quantized Model	99, 100	99, 100	99.75, 100	0.75, 100
	Versal Model	99.5,100	99.5, 100	99.75, 100	79.95, 100
ResNet56	Quantized Model	99.75, 100	99.5, 100	99.75, 100	0.25, 100
	Versal Model	99.75, 100	99.75, 100	99.75, 100	87.47, 100
MobileNetV2	Quantized Model	99.75, 100	99.75, 100	99.75, 100	0, 100
	Versal Model	100, 100	100, 100	100, 100	95.74, 100

Table 6.29: PSNR Evaluation adversarial train accuracy

changes on our part. From this phenomenon along with the results of the CIFAR-10 dataset we can safely conclude that our deployed models feature a similar if not increased adversarial robustness with our quantized ones effectively completing our objectives and the scope of this thesis.

Chapter 7

Conclusion

Summarizing our work, we effectively applied a three-step adversarial training process along with quantization on ResNet20, ResNet56 and MobileNetV2 models in CIFAR-10 and FashionMNIST, achieving significant adversarial robustness improvements. Furthermore, we optimized these models for resource-constrained hardware deployment preserving their robustness and increasing their accuracy. All in all, through our proposed methodology and utilization of the Versal Platform we not only accomplish a total PSNR decrease of ≈40% but also with minimal accuracy losses of <0.25% for each dataset. Specifically, for CIFAR-10 we achieve a PSNR decrease of 42% in ResNet20, 43% in ResNet56 and 45% in MobileNetV2 with our final PSNR ranges at 31-32 dB. For FashionMNIST, we achieve 37%, 39% and 45% PSNR decrease for each model respectively with the final PSNR ranges at 24-29 dB. With PSNR being a noise evaluation metric, these values signify the existence of heavy amounts of noise in each image. Thus extracting and evaluating the images we reach a point where in order for our models to misclassify them, heavy perturbations easily detectable by humans must be introduced. To conclude, these low PSNR values directly translate to increased adversarial robustness proving the effectiveness of our methodologies and producing robust accurate models for hardware deployment.

7.1 Future research

This work while providing promising results, can be further optimized for its efficiency, effectiveness and scalability. Through these refinements we aim to provide an even more robust and scalable solution for all heterogenous hardware devices.

- Improved Robustness by applying preprocessing algorithms and hardware specific techniques.
- Use of another, similarly reliable to HSJA, attack reducing time constraints.
- Integration of hardware defense methods enhancing the overall security of neural networks.
- Expansion to other image classification architectures.

7.2 Challenges

Achieving these results while simple in practice, highlights the significant drawback of utilizing black-box algorithms for adversarial examples. While they provide excellent performance for our defense methods, their black-box nature of repeatedly querying the model leads to extreme time requirements for image generation. These requirements scale accordingly to the complexity of architectures with ResNet56 for instance requiring double the time of ResNe20 to generate an example. Since we generate examples on both quantized and floating-point (fp) versions, we are able to utilize powerful GPUs for the floating point variants and CPUs for the quantized ones. However, even with these resources, approximately 6.5 seconds and 18 seconds are required for one image to be generated on ResNet20 in fp and quantized variants. These time requirements only increase with ResNet56 and MobileNetV2 requiring approximately 14 seconds and 25 seconds for their floating point and quantized variants. Given that we generate 60000 examples per model per dataset for our training purposes and another 1600 on the floating point and quantized versions for evaluations, the total time for each dataset comes to 119 hours for ResNet20 and 250 hours for ResNet56 and MobileNetV2 respectively. Additionally, these estimations do not include the training and evaluation or creation of the code rather they only account for the continuous computational requirements to generate the examples.

Bibliography

[1] Google colaboratory. URL `https://colab.research.google.com`.

[2] Kaggle. URL `https://www.kaggle.com`.

[3] Peak signal-to-noise ration example images. URL `https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio`.

[4] Pytorch framework. URL `https://pytorch.org`.

[5] Adversarial robustness toolbox. GitHub. URL `https://github.com/Trusted-AI/adversarial-robustness-toolbox`.

[6] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018. doi: 10.1109/ACCESS.2018.2807385. URL `https://ieeexplore.ieee.org/abstract/document/8294186`.

[7] Mert Alagözlü. *Stochastic Gradient Descent Variants and Applications*. 2022. doi: 10.13140/RG.2.2.12528.53767. URL `https://doi.org/10.13140/RG.2.2.12528.53767`.

[8] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples, 2018. URL `https://arxiv.org/abs/1804.07998`.

[9] Xilinx AMD. Versal: The first adaptive compute acceleration platform, 2020. URL `https://docs.xilinx.com/v/u/en-US/wp505-versal-acap`.

[10] Xilinx AMD. Vck190 evaluation board user guide, 2023. URL `https://docs.xilinx.com/r/en-US/ug1366-vck190-eval-bd/Board-Features`.

[11] Xilinx AMD. Versal™ architecture and product data sheet, 2024. URL `https://docs.xilinx.com/v/u/en-US/ds950-versal-overview`.

[12] Janez Bester Andrej Krenker and Andrej Kos. *Introduction to the Artificial Neural Networks.* InTech, 2011. doi: 10.5772/15751. URL `https://www.intechopen.com/chapters/14881`.

[13] Bo Chen Andrew G. Howard, Menglong Zhu et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* arXiv, 2017. doi: 1704.04861. URL `https://arxiv.org/abs/1704.04861`.

[14] Niki Parmar Ashish Vaswani, Noam Shazeer et al. *Attention Is All You Need.* arXiv, 2017. doi: 1706.03762. URL `https://doi.org/10.48550/arXiv.1706.03762`.

[15] Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems.* Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/36d7534290610d9b7e9abed244dd2f28-Abstract.html`.

[16] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1): 25–45, 2021. doi: https://doi.org/10.1049/cit2.12028. URL `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12028`.

[17] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack, 2020. URL `https://arxiv.org/abs/1904.02144`.

[18] chenyaofo. Pytorch cifar models. GitHub, 2021. URL `https://github.com/chenyaofo/pytorch-cifar-models`.

[19] Stanford CS. *CS231n: Deep Learning for Computer Vision.* URL `https://cs231n.github.io/convolutional-networks/`.

[20] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data, 2018. URL `https://arxiv.org/abs/1806.02371`.

[21] Ke Dong, Chengjie Zhou, Yihan Ruan, and Yuzhi Li. Mobilenetv2 model for image classification. In *2020 2nd International Conference on Information Technology and Computer Application (ITCA).* doi: 10.1109/ITCA52113.2020.00106. URL `https://ieeexplore.ieee.org/abstract/document/9422058`.

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL `http://jmlr.org/papers/v12/duchi11a.html`.

[23] Kirsty Duncan, Ekaterina Komendantskaya, Robert Stewart, and Michael Lones. Relative robustness of quantized neural networks against adversarial attacks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9207596.

[24] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification, 2018. URL `https://arxiv.org/abs/1712.06751`.

[25] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021. URL `https://arxiv.org/abs/2103.13630`.

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples.* arXiv, 2015. doi: 1412.6572. URL `https://arxiv.org/abs/1412.6572`.

[27] Micah Gorsline, James Smith, and Cory Merkel. On the adversarial robustness of quantized neural networks. GLSVLSI '21. Association for Computing Machinery, 2021. ISBN 9781450383936. doi: 10.1145/3453688.3461755. URL `https://doi.org/10.1145/3453688.3461755`.

[28] Mahowald MA Hahnloser RH, Sarpeshkar R et al. *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit.* Nature, 2000. doi: 10.1038/35016072. URL `https://doi.org/10.1038/35016072`.

[29] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. Towards security threats of deep learning systems: A survey. *IEEE Transactions on Software Engineering*, 48(5):1743–1770, 2022. doi: 10.1109/TSE.2020.3034721. URL `https://ieeexplore.ieee.org/abstract/document/9252914`.

[30] Srivastava N. Hinton G. and Swerski K. Lecture 6d - a separate, adaptive learning rate for each connecton. Slides of Lecture Neural Networks for Machine Learning, 2012.

[31] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvari. Learning with a strong adversary, 2016.

[32] Mehdi Mirza Ian Goodfellow, Jean Pouget-Abadie et al. *Generative Adversarial Nets.* Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`.

[33] Yunseok Jang, Tianchen Zhao, Seunghoon Hong, and Honglak Lee. Adversarial defense via learning to generate diverse attacks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2740–2749, 2019. doi: 10.1109/ICCV.2019.00283.

[34] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv, 2015. doi: 1512.03385. URL `https://arxiv.org/abs/1512.03385`.

[35] Ryan Nash Keiron O'Shea. *An Introduction to Convolutional Neural Networks*. arXiv, 2015. doi: 10.48550/arXiv.1511.08458. URL `https://doi.org/10.48550/arXiv.1511.08458`.

[36] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv, 2017. doi: 1412.6980. URL `https://doi.org/10.48550/arXiv.1412.6980`.

[37] Oscar Knagg, 2019. URL `https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3`.

[38] AAlex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[39] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017. URL `https://arxiv.org/abs/1611.01236`.

[40] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2017.

[41] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Transactions on Dependable and Secure Computing*, 18(1):72–85, January 2021. ISSN 2160-9209. doi: 10.1109/tdsc.2018.2874243. URL `http://dx.doi.org/10.1109/TDSC.2018.2874243`.

[42] Haowen Lin, Jian Lou, Li Xiong, and Cyrus Shahabi. Integer-arithmetic-only certified robustness for quantized neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7828–7837, October 2021.

[43] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019. URL `https://arxiv.org/abs/1706.06083`.

[44] Menglong Zhu Mark Sandler, Andrew Howard et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv, 2019. doi: 1801.04381. URL `https://arxiv.org/abs/1801.04381v4`.

[45] Chaithanya Kumar Mummadi, Thomas Brox, and Jan Hendrik Metzen. Defending against universal perturbations with shared adversarial training, 2019.

[46] Nicolas Papernot and Patrick McDaniel. On the effectiveness of defensive distillation, 2016.

[47] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2016.

[48] Nicolas Papernot, Patrick McDaniel, Goodfellow, et al. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349444. doi: 10.1145/3052973.3053009. URL `https://doi.org/10.1145/3052973.3053009`.

[49] Jürgen Schmidhuber Sepp Hochreiter. *Long Short-Term Memory.* MIT Press, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[50] Chang Song, Elias Fallon, and Hai Li. Improving adversarial robustness in weight-quantized neural networks, 2021.

[51] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data, 2022.

[52] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. *Intriguing properties of neural networks.* arXiv, 2014. doi: 1312.6199. URL `https://arxiv.org/abs/1312.6199`.

[53] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training, 2020.

[54] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[55] Xilinx. Vitisai, . URL `https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html`.

[56] Xilinx. Versal vck190, . URL `https://www.xilinx.com/products/boards-and-kits/vck190.html`.

[57] Xilinx. Introduction to fpga design with vivado high-level synthesis (ug998), 2019. URL `https://docs.xilinx.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls`.

[58] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 2020. doi: 10.1007/s11633-019-1211-x. URL `https://doi.org/10.1007/s11633-019-1211-x`.

[59] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings 2018 Network and Distributed System Security Symposium*, NDSS 2018. Internet Society, 2018. doi: 10.14722/ndss.2018.23198. URL `http://dx.doi.org/10.14722/ndss.2018.23198`.

[60] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Me-net: Towards effective adversarial robustness with matrix estimation, 2019.

[61] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.