Διεθνές Πανεπιστήμιο της Ελλάδος
Τμήμα Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών

Υλοποίηση αλγορίθμων μηχανικής μάθησης με χρήση FPGA

Παναγιώτου Νικόλαος

Διεθνές Πανεπιστήμιο της Ελλάδος

Σέρρες, 11-09-2023

Εργασία που υποβλήθηκε στο
Πρόγραμμα Μεταπτυχιακών Σπουδών στη Ρομποτική,
του Διεθνούς Πανεπιστημίου της Ελλάδος,
για τη μερική εκπλήρωση υποχρεώσεων για το Δίπλωμα Ειδίκευσης στη Ρομποτική

Επιβλέπων Καθηγητής: Βουρβουλάκης Ιωάννης

Περίληψη

Η παρούσα διπλωματική διερευνά την υλοποίηση τεχνητών νευρωνικών δικτύων σε γλώσσα προγραμματισμού VHDL. Παρουσιάζεται το πώς οι βασικές λειτουργίες ενός νευρώνα - όπως η σταθμισμένη άθροιση και η ενεργοποίηση - μπορούν να πραγματοποιηθούν αποτελεσματικά σε ολοκληρωμένα κυκλώματα FPGA. Ο σχεδιασμός εστιάζει στη γενικότητα των βαρών και του bias και αξιοποιεί μια προσέγγιση υλοποίησης του νευρωνικού μέσω του Octave.

Επιπλέον εμβαθύνει στον σχεδιασμό σε VHDL ενός νευρωνικού δικτύου προσαρμοσμένου για επεξεργασία εικόνας RGB. Αξιοποιώντας μια αρχιτεκτονική πολλαπλών επιπέδων perceptron (MLP), αυτή η μονάδα VHDL στοχεύει στην επεξεργασία εισερχόμενων σημάτων εικονοστοιχείων από μια εικόνα και πιο συγκεκριμένα τα στοιχεία RGB του κάθε pixel, ώστε να δημιουργήσει μια έξοδο όπου τα pixel αυτά δημιουργούν μια νέα εικόνα.

Τέλος μέσω της βιβλιοθήκης KERAS εκπαιδεύεται έναν νευρωνικό δίκτυο σε γλώσσα Python για την αναγνώριση χειρόγραφων αριθμών μέσα από ένα πλήθος εικόνων.

Abstract

This thesis explores the implementation of artificial neural networks in VHDL programming language. It shows how the basic functions of a neuron - such as weighted summation and activation - can be efficiently implemented on FPGA boards. The design focuses on the generality of weights and bias and leverages a neural implementation approach through Octave.

It further delves into the design in VHDL of a custom neural network for RGB image processing. Leveraging a multi-layer perceptron (MLP) architecture, this VHDL module aims to process incoming image signals from an image, and more specifically RGB components of each pixel, to produce an output where those pixels form a new image.

Finally, through the KERAS library, a neural network in Python language is trained to recognize handwritten numbers from a multitude of images.

Περιεχόμενα

1. Εισαγωγή

Τα τελευταία χρόνια, ο τομέας της μηχανικής μάθησης έχει σημειώσει τεράστια πρόοδο, επιτρέποντας καινοτόμες εφαρμογές σε τομείς όπως η όραση υπολογιστών [1], η επεξεργασία φυσικής γλώσσας [2] και τα αυτόνομα συστήματα [3]. Αυτή η πρόοδος έχει οδηγήσει σε αυξανόμενη ζήτηση για υπολογιστική ισχύ και ενεργειακή απόδοση. Λύση σε αυτή την ανάγκη έρχονται να δώσουν τα FPGA. Τα FPGA προσφέρουν την ευελιξία του υλικού που ορίζεται από λογισμικό, καθιστώντας τα ελκυστική επιλογή για την εφαρμογή αλγορίθμων μηχανικής μάθησης.

Με τη συνεχή ανάπτυξη των ψηφιακών δεδομένων και την ανάγκη για επεξεργασία σε πραγματικό χρόνο, ο τομέας της τεχνητής νοημοσύνης (AI) και, πιο συγκεκριμένα, των νευρωνικών δικτύων (NNs) έχει έρθει στο προσκήνιο της τεχνολογικής προόδου. Τα Νευρωνικά δίκτυα είναι μαθηματικά μοντέλα εμπνευσμένα από τη δομή του ανθρώπινου εγκεφάλου, ικανά να αναγνωρίζουν μοτίβα, να κάνουν προβλέψεις και να μαθαίνουν από δεδομένα. Παραδοσιακά, αυτοί οι υπολογισμοί πραγματοποιούνται σε κεντρικές μονάδες επεξεργασίας (CPU) και, πιο πρόσφατα, σε μονάδες επεξεργασίας γραφικών (GPU).[4] Ωστόσο, καθώς η ζήτηση για ταχύτερους και πιο ενεργειακά αποδοτικούς υπολογισμούς έχει αυξηθεί, έχει αυξηθεί το ενδιαφέρον για την εξερεύνηση εναλλακτικών πλατφορμών υλικού. Μια τέτοια πλατφόρμα που προσφέρει πολλά υποσχόμενες δυνατότητες είναι η Field-Programmable Gate Array (FPGA).

Τα FPGA είναι ολοκληρωμένα κυκλώματα που μπορούν να επαναπρογραμματιστούν σε επιθυμητές εφαρμογές μετά την κατασκευή, προσφέροντας σημαντικά πλεονεκτήματα όσον αφορά την ευελιξία, την ταχύτητα και την απόδοση ισχύος σε σχέση με τα παραδοσιακά συστήματα υπολογιστών. Η εγγενώς παράλληλη αρχιτεκτονική τους και η δυνατότητα προσαρμογής τους τα καθιστούν ιδιαίτερα κατάλληλα για την εφαρμογή αλγορίθμων νευρωνικών δικτύων. Προσαρμόζοντας το υλικό FPGA στις συγκεκριμένες απαιτήσεις των αλγορίθμων νευρωνικών δικτύων, είναι δυνατό να επιτευχθούν επίπεδα απόδοσης και αποδοτικότητας που μπορούν να συγκριθούν και να ξεπεράσουν τα αποτελέσματα των επεξεργαστών γενικής χρήσης.[5]

Αυτή η διατριβή στοχεύει να εμβαθύνει στα ζητήματα της εφαρμογής νευρωνικών δικτύων σε FPGA, διερευνώντας τα πλεονεκτήματα και τις μεθοδολογίες που εμπλέκονται.

1.1. FPGA και ιστορική αναδρομή

Οι Field Programmable Gate Arrays (FPGAs) είναι διατάξεις ημιαγωγών που βασίζονται γύρω από συστοιχίες προγραμματιζόμενων λογικών μπλοκ (CLBs) και συνδέονται μέσω προγραμματιζόμενων διασυνδέσεων. Τα FPGA είναι ολοκληρωμένα κυκλώματα σχεδιασμένα να διαμορφώνονται μετά την κατασκευή στις απαιτήσεις του εκάστοτε χρήση-προγραμματιστή. Ένας τρόπος για να καθοριστεί το κύκλωμα που θα φορτωθεί σε ένα FPGA είναι η χρήση γλώσσας περιγραφής υλικού (HDL). Τα FPGA περιέχουν μια σειρά από προγραμματιζόμενα λογικά μπλοκ και μια ιεραρχία διασυνδέσεων που μπορούν να διαμορφωθούν ξανά και ξανά, και επιτρέπουν στα μπλοκ να συνδεθούν μεταξύ τους. Τα λογικά μπλοκ μπορούν να διαμορφωθούν για να εκτελούν σύνθετες συνδυαστικές λειτουργίες ή να λειτουργούν ως απλές λογικές πύλες. Στα περισσότερα FPGA. τα λογικά μπλοκ περιλαμβάνουν επίσης στοιχεία μνήμης, τα οποία μπορεί να είναι απλά flip-flops ή πιο ολοκληρωμένα μπλοκ μνήμης. Τα FPGA διαδραματίζουν σημαντικό ρόλο στην ανάπτυξη ενός ενσωματωμένου συστήματος λόγω της ικανότητάς τους να ξεκινούν την ανάπτυξη λογισμικού συστήματος (SW) ταυτόχρονα με το υλικό (HW), να επιτρέπουν προσομοιώσεις απόδοσης συστήματος σε πολύ πρώιμο στάδιο της ανάπτυξης και να επιτρέπουν διάφορες διαμερίσεις συστήματος (SW και HW) δοκιμές και επαναλήψεις πριν από την τελική κατάσταση της αρχιτεκτονικής του συστήματος. [5]

Η βιομηχανία των FPGA αναπτύχθηκε από προγραμματιζόμενη μνήμη μόνο για ανάγνωση (PROM) και προγραμματιζόμενες συσκευές λογικής (PLD). Τα PROM και τα PLD είχαν και τα δύο τη δυνατότητα να προγραμματιστούν σε παρτίδες σε ένα εργοστάσιο ή στο πεδίο (προγραμματιζόμενο σε πεδίο). Ωστόσο, η προγραμματιζόμενη λογική ήταν συνδεδεμένη μεταξύ των λογικών πυλών.

Η Altera ιδρύθηκε το 1983 και παρέδωσε την πρώτη επαναπρογραμματιζόμενη λογική συσκευή της βιομηχανίας το 1984 – την EP300 – η οποία περιείχε ένα παράθυρο από χαλαζία στη συσκευασία που επέτρεπε στους χρήστες να βάλουν μια λάμπα υπεριώδους στο καλούπι για να διαγράψουν τα κύτταρα EPROM που κρατούσαν τη διαμόρφωση της συσκευής .

Οι συνιδρυτές της Xilinx, Ross Freeman και Bernard Vonderschmitt, ανακάλυψαν την πρώτη εμπορικά βιώσιμη διάταξη πυλών προγραμματιζόμενης πεδίου το 1985 – το XC2064. Το XC2064 είχε προγραμματιζόμενες πύλες και προγραμματιζόμενες διασυνδέσεις μεταξύ των πυλών, την αρχή μιας νέας τεχνολογίας και αγοράς. Το XC2064 είχε 64 ρυθμιζόμενα λογικά μπλοκ (CLB), με δύο πίνακες αναζήτησης τριών

εισόδων (LUT). Περισσότερα από 20 χρόνια αργότερα, ο Freeman μπήκε στο National Inventors Hall of Fame για την εφεύρεσή του.

Το 1987, το Naval Surface Warfare Center χρηματοδότησε ένα πείραμα που πρότεινε ο Steve Casselman για την ανάπτυξη ενός υπολογιστή που θα υλοποιούσε 600.000 επαναπρογραμματιζόμενες πύλες. Ο Casselman ήταν επιτυχής με αποτέλεσμα το 1992 να εκδοθεί το δίπλωμα ευρεσιτεχνίας που σχετίζεται με το σύστημα.

Η Altera και η Xilinx συνέχισαν χωρίς αμφισβήτηση και αναπτύχθηκαν γρήγορα από το 1985 έως τα μέσα της δεκαετίας του 1990, όταν αναπτύχθηκε ανταγωνισμός, κερδίζοντας σημαντικό μεριδίου αγοράς τους. Μέχρι το 1993, η Actel (μετέπειτα Microsemi και τώρα Microchip) εξυπηρετούσε περίπου το 18 τοις εκατό της αγοράς.

Η δεκαετία του 1990 ήταν μια περίοδος ταχείας ανάπτυξης για τα FPGA, τόσο στην πολυπλοκότητα του κυκλώματος όσο και στον όγκο της παραγωγής. Στις αρχές της δεκαετίας του 1990, τα FPGA χρησιμοποιήθηκαν κυρίως στις τηλεπικοινωνίες και τη δικτύωση. Μέχρι το τέλος της δεκαετίας, τα FPGA βρήκαν το δρόμο τους σε καταναλωτικές, αυτοκινητοβιομηχανίες και βιομηχανικές εφαρμογές.

Μέχρι το 2013, η Altera (31 τοις εκατό), η Actel (10 τοις εκατό) και η Xilinx (36 τοις εκατό) αντιπροσώπευαν μαζί περίπου το 77 τοις εκατό της αγοράς FPGA.

Εταιρείες όπως η Microsoft έχουν αρχίσει να χρησιμοποιούν FPGA για να επιταχύνουν συστήματα υψηλής απόδοσης και υπολογιστικής έντασης (όπως τα κέντρα δεδομένων που λειτουργούν τη μηχανή αναζήτησής τους Bing), λόγω του πλεονεκτήματος απόδοσης ανά watt που παρέχουν τα FPGA. Η Microsoft άρχισε να χρησιμοποιεί FPGA για να επιταχύνει το Bing το 2014 και το 2018 άρχισε να χρησιμοποιεί FPGA σε άλλους φόρτους εργασιών κέντρων δεδομένων για την πλατφόρμα υπολογιστών cloud Azure. [5]

1.2. Μηχανική Μάθηση

Ο άνθρωπος προσπαθεί να κατανοήσει τον κόσμο περισσότερο χρησιμοποιώντας τις παρατηρήσεις και τις εμπειρίες του. Για να το κάνει αυτό, δημιουργεί απλοποιημένες εκδοχές του κόσμου, που τις ονομάζουμε "μοντέλα." Η διαδικασία αυτή ονομάζεται επαγωγική μάθηση (inductive learning). Επίσης, μπορεί να οργανώσει τις εμπειρίες του δημιουργώντας πρότυπα, που είναι νέες δομές που βοηθούν στην κατανόηση του κόσμου.

Η δημιουργία μοντέλων ή προτύπων από ένα σύνολο δεδομένων, από ένα

υπολογιστικό σύστημα, δηλαδή όταν μια μηχανή προσπαθεί να κατανοήσει τον κόσμο χρησιμοποιώντας δεδομένα, ονομάζεται "μηχανική μάθηση" (machine learning).

Διάφοροι ορισμοί:

- Carbonell (1987), "... η μελέτη υπολογιστικών μεθόδων για την απόκτηση νέας γνώσης, νέων δεξιοτήτων και νέων τρόπων οργάνωσης της υπάρχουσας γνώσης".
- Mitchell (1997), "Ένα πρόγραμμα υπολογιστή θεωρείται ότι μαθαίνει από την εμπειρία E σε σχέση με μια κατηγορία εργασιών T και μια μετρική απόδοσης P, αν η απόδοση του σε εργασίες της T, όπως μετριούνται από την P, βελτιώνονται με την εμπειρία E".
- Witten & Frank (2000), "Κάτι μαθαίνει όταν αλλάζει τη συμπεριφορά του κατά τέτοιο τρόπο ώστε να αποδίδει καλύτερα στο μέλλον".

Έχουν αναπτυχθεί πολλές τεχνικές μηχανικής μάθησης που χρησιμοποιούνται ανάλογα με τη φύση του προβλήματος και εμπίπτουν σε ένα από τα παρακάτω δυο είδη:

- μάθηση με επίβλεψη (supervised learning) ή μάθηση με παραδείγματα (learning from examples),
- μάθηση χωρίς επίβλεψη (unsupervised learning) ή μάθηση από παρατήρηση (learning from observation).

Στη "μάθηση με επίβλεψη," η μηχανή προσπαθεί να μάθει μια έννοια ή μια συνάρτηση από ένα σύνολο παραδειγμάτων. Για παράδειγμα, μπορεί να προσπαθεί να μάθει να αναγνωρίζει φωτογραφίες γατών από ένα σύνολο φωτογραφιών που έχουν ετικεταριστεί ως "γάτες" ή "όχι γάτες". Στη μάθηση χωρίς επίβλεψη το σύστημα πρέπει μόνο του να ανακαλύψει συσχετίσεις ή ομάδες σε ένα σύνολο δεδομένων, δημιουργώντας πρότυπα, χωρίς να είναι γνωστό αν υπάρχουν, πόσα και ποια είναι. [6]

Σε γενικές γραμμές, η μηχανική μάθηση είναι ο τρόπος με τον οποίο οι μηχανές μπορούν να μάθουν και να προσαρμόζονται στον κόσμο.

1.3. Εφαρμογές ML σε FPGA

Accelerated Inference: Τα FPGA μπορούν να χρησιμοποιηθούν για την επιτάχυνση της φάσης συμπερασμάτων των μοντέλων μηχανικής εκμάθησης. Αυτό είναι ιδιαίτερα χρήσιμο σε εφαρμογές σε πραγματικό χρόνο, όπως η ανίχνευση αντικειμένων, η αναγνώριση εικόνας και η επεξεργασία φυσικής γλώσσας. Με τη χρήση FPGA, είναι εφικτό να επιτευχθεί επεξεργασία υψηλής απόδοσης με πολύ μικρό χρόνο καθυστέρησης. [7]

Custom Hardware Accelerators: Τα FPGA επιτρέπουν τη δημιουργία προσαρμοσμένων επιταχυντών υλικού σε συγκεκριμένους αλγόριθμους ML. Αυτό μπορεί να οδηγήσει σε σημαντική επιτάχυνση σε σύγκριση με τις παραδοσιακές υλοποιήσεις CPU ή GPU. Για παράδειγμα, είναι εφικτό να σχεδιαστεί custom hardware για συνελικτικά Νευρωνικά δίκτυα (CNN), επαναλαμβανόμενα Νευρωνικά δίκτυα (RNN) ή άλλα εξειδικευμένα μοντέλα. [6]

Edge AI: Τα FPGA είναι κατάλληλα για εφαρμογές άκρων τεχνητής νοημοσύνης όπου η απόδοση ισχύος και η επεξεργασία σε πραγματικό χρόνο είναι ζωτικής σημασίας. Μπορούν να ενσωματωθούν σε συσκευές αιχμής όπως κάμερες, drones, ρομπότ και συσκευές IoT για την εκτέλεση εργασιών ML χωρίς να βασίζονται στο cloud. Αυτό είναι απαραίτητο για το απόρρητο και τη μείωση της καθυστέρηση. [8]

High-Performing Computing: Τα FPGA μπορούν να χρησιμοποιηθούν σε περιβάλλοντα υπολογιστών υψηλής απόδοσης (HPC) για την επιτάχυνση των επιστημονικών προσομοιώσεων, της ανάλυσης δεδομένων και της εκπαίδευσης των Deep Neural Networks (βαθιά Νευρωνικά Δίκτυα). Οι προσαρμοσμένοι επιταχυντές υλικού που έχουν σχεδιαστεί για συγκεκριμένους φόρτους εργασίας HPC μπορούν να βελτιώσουν σημαντικά την απόδοση. [9]

Ασφάλεια και κρυπτογράφηση: Η ML σε FPGA μπορεί να βελτιώσει τις εφαρμογές ασφαλείας επιταχύνοντας κρυπτογραφικούς αλγόριθμους και συστήματα ανίχνευσης εισβολών. Τα FPGA μπορούν να προγραμματιστούν ώστε να εφαρμόζουν πολύπλοκες διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης, καθιστώντας τα κατάλληλα για εφαρμογές ασφάλειας δικτύου και προστασίας δεδομένων. [10]

Χρηματοοικονομική μοντελοποίηση: Στον χρηματοπιστωτικό κλάδο, τα FPGA χρησιμοποιούνται για την αξιολόγηση κινδύνου, τους αλγόριθμους συναλλαγών και τις συναλλαγές υψηλής συχνότητας. Οι προσαρμοσμένοι επιταχυντές υλικού μπορούν να υπολογίσουν πολύπλοκα οικονομικά μοντέλα και προσομοιώσεις ταχύτερα από τις παραδοσιακές CPU. [11]

Ιατρική Απεικόνιση: Τα FPGA χρησιμοποιούνται σε εφαρμογές ιατρικής απεικόνισης, συμπεριλαμβανομένης της ανακατασκευής εικόνων MRI και CT, καθώς και ανάλυση ιατρικών δεδομένων σε πραγματικό χρόνο. Μπορούν να παρέχουν την απαιτούμενη υπολογιστική ισχύ διατηρώντας παράλληλα χαμηλή κατανάλωση ενέργειας. [12]

Επεξεργασία βίντεο και εικόνας: Τα FPGA χρησιμοποιούνται για επεξεργασία βίντεο και εικόνας σε πραγματικό χρόνο σε εφαρμογές όπως η παρακολούθηση βίντεο, τα συστήματα όρασης αυτοκινήτου και η επαυξημένη πραγματικότητα. Μπορούν να επιταχύνουν εργασίες όπως φιλτράρισμα εικόνας, εξαγωγή χαρακτηριστικών και παρακολούθηση αντικειμένων. [13]

Επεξεργασία ομιλίας και φυσικής γλώσσας: Τα FPGA μπορούν να χρησιμοποιηθούν για αναγνώριση ομιλίας σε πραγματικό χρόνο, μετάφραση γλώσσας και ανάλυση συναισθήματος. Οι προσαρμοσμένοι επιταχυντές μπορούν να επεξεργάζονται δεδομένα ήχου και κειμένου αποτελεσματικά. [14]

Ρομποτική: Τα FPGA είναι ενσωματωμένα σε ρομπότ για να επιταχύνουν εργασίες αντίληψης όπως η ανίχνευση αντικειμένων και ο εντοπισμός αντικειμένων. Αυτό δίνει τη δυνατότητα στα ρομπότ να λαμβάνουν πιο γρήγορες και ακριβείς αποφάσεις, ενισχύοντας την αυτονομία και την αποτελεσματικότητά τους. [15]

2. Κατάσταση της Τέχνης

2.1. Διατύπωση του προβλήματος

Η παρούσα εργασία έχει σκοπό την μελέτη, την ανάλυση και την υλοποίηση των εφαρμογών αλγορίθμων μηχανικής μάθησης σε FPGA και περιλαμβάνει δύο διαφορετικές προσεγγίσεις για την ανάπτυξη νευρωνικών δικτύων για τον εντοπισμό εικόνων σημάτων οδικής κυκλοφορίας.

- Ανάπτυξη νευρωνικού δικτύου για τον εντοπισμό αριθμών μέσα από ένα σύνολο εικόνων που απεικονίζουν χειρόγραφους αριθμούς με την χρήση του Jupiter Notebook και της βιβλιοθήκη hls4ml
- Ανάπτυξη νευρωνικού δικτύου για τον εντοπισμό εικόνων σημάτων οδικής κυκλοφορίας σε γλώσσα VHDL

Αξιοποιήθηκαν δύο διαφορετικοί τρόποι προσέγγισης του προβλήματος. Η πρώτη προσέγγιση χρησιμοποιεί τη γλώσσα προγραμματισμού Python και το περιβάλλον Jupyter Notebook για τη σχεδίαση νευρωνικών δικτύων. Η βιβλιοθήκη hls4ml χρησιμοποιείται στη μετατροπή του Νευρωνικού Δικτύου σε κώδικα VHDL. Αυτή η προσέγγιση είναι πιο ευέλικτη και απλοποιεί τη διαδικασία ανάπτυξης μοντέλων μηχανικής μάθησης, καθώς χρησιμοποιεί μια γλώσσα προγραμματισμού υψηλού επιπέδου και προσφέρει εργαλεία για ευκολότερη ανάπτυξη και δοκιμή.

Σύμφωνα με την δεύτερη προσέγγιση, σχεδιάστηκε ένα νευρωνικό δίκτυο στο Octave. Στη συνέχεια αναπτύχθηκε κώδικας που υλοποιεί το νευρωνικό δίκτυο σε γλώσσα περιγραφής υλικού VHDL. Αυτή η προσέγγιση είναι κατάλληλη για περιπτώσεις όπου η υλοποίηση σε υλικό (FPGA) απαιτείται για υψηλή απόδοση και χαμηλή καθυστέρηση.

2.2. Κατάσταση της Τέχνης (State of the art)

Η υλοποίηση αλγορίθμων της μηχανικής μάθησης (ML) στις Field Programmable Gate Arrays (FPGA) έχει σημειώσει αξιοσημείωτη πρόοδο τα τελευταία χρόνια. Τα FPGA κερδίζουν εξέχουσα θέση στο αντικείμενο της μηχανικής μάθησης λόγω του εγγενούς παραλληλισμού τους, των δυνατοτήτων low-latency και της ενεργειακής τους απόδοσης. Ερευνητές και μηχανικοί αξιοποιούν την τεχνολογία FPGA για να επιταχύνουν ένα ευρύ φάσμα φόρτων εργασίας ML, από τη βαθιά μάθηση έως τις εφαρμογές προσαρμοσμένων αλγορίθμων. Η υψηλή απόδοση που παρέχεται από τα FPGA επιτρέπουν την ανάπτυξη μοντέλων ML σε συσκευές άκρων και κέντρα δεδομένων, καθιστώντας εφικτή την επεξεργασία σε πραγματικό χρόνο και την χαμηλή κατανάλωση. Επιπλέον, οι εταιρείες κατασκευής FPGA αναπτύσσουν εξειδικευμένες πλατφόρμες και kit εργαλείων, διευκολύνοντας τους προγραμματιστές να αξιοποιήσουν τις δυνατότητες των FPGA σε διάφορες εφαρμογές. Ως αποτέλεσμα, η χρήση των FPGA σε εφαρμογές της ML συνεχίζει να εξελίσσεται, προσφέροντας την προοπτική για ταχύτερες, πιο ενεργειακά αποδοτικές και ευέλικτες υλοποιήσεις εφαρμογών ML. [16][17][18]

Αρκετά αξιοσημείωτα παραδείγματα παρουσιάζουν τις σύγχρονες εφαρμογές της ML σε FPGA μερικά εκ των οποίων είναι τα άρθρα που θα αναλυθούν παρακάτω.

Πρώτο παράδειγμα που επιβεβαιώνει την εφαρμογή της ML σε FPGA αποτελεί η δημοσίευση τον Σεπτέμβριο του 2020 του άρθρο “FPGA-Based Network Traffic Classification Using Machine Learning”. Το παρόν άρθρο είναι αποτέλεσμα επιστημονικής έρευνας και ανάλυσης των Mohammed Elnawawy , Assim Sagahyroon και Tamer Shanableh. Ανέλυσαν το πώς μπορεί να ταξινομηθεί η «κίνηση» των πακέτων μέσα σε ένα σύνολο δικτύων με την χρήση ενός FPGA. Στο συγκεκριμένο άρθρο οι συγγραφείς χρησιμοποίησαν τα αρχικά πακέτα πλήθους ‘n’ μέσα σε μια ροή ενός δικτύου. Ενώ οι υψηλότερες τιμές του ‘n’ βελτιώνουν το classification εισάγουν καθυστερήσεις, αντίθετα οι μικρότερες τιμές του ‘n’ έχουν ως αποτέλεσμα λιγότερη ακρίβεια στην ταξινόμηση. Σε αυτήν την έρευνα, προσδιορίζεται μια κατάλληλη τιμή για τον αριθμό των πακέτων που επιτυγχάνει μια ισορροπία μεταξύ της ακρίβειας του classification και της καθυστέρησης. Επιπλέον, διερευνά τον αντίκτυπο της ενσωμάτωσης πρόσθετων χαρακτηριστικών σε επίπεδο ροής στην απόδοση ταξινόμησης. Ακόμα, αξίζει να σημειωθεί ότι πολλές μελέτες σε αυτόν τον τομέα επικεντρώνονται στην ανάπτυξη classifiers που βασίζονται σε port numbers τον αριθμό των θυρών δηλαδή, οι οποίες έχουν γίνει λιγότερο αξιόπιστες για την ταξινόμηση της

κυκλοφορίας καθώς οι σύγχρονες εφαρμογές συχνά αλλάζουν τη χρήση των θυρών τους για να αποφύγουν τον εντοπισμό. Κατά συνέπεια, παρουσιάζεται ένας αποτελεσματικός ταξινομητής που δεν εξαρτάται από τον αριθμό των θυρών. Όσον αφορά την εφαρμογή ταξινομητών κυκλοφορίας που βασίζονται σε λογισμικό, προσπαθούν να χειριστούν τον τεράστιο όγκο της κυκλοφορίας δικτύου, γεγονός που καθιστά αναγκαία την υιοθέτηση ταξινομητών κυκλοφορίας με επιτάχυνση υλικού. Επομένως, αυτό το άρθρο προτείνει έναν ταξινομητή κίνησης δικτύου που βασίζεται σε υλικό που χρησιμοποιεί random forest classifier σε μια πλατφόρμα FPGA.

Επόμενο άρθρο είναι των Shen Zhang, Oliver Wallscheid και Mario Porrmann με τίτλο “Machine Learning for the Control and Monitoring of Electric Machine Drives: Advances and Trend”. Αυτό το άρθρο ερευνά τη χρήση τεχνικών μηχανικής εκμάθησης (ML) για τον έλεγχο και την παρακολούθηση των ηλεκτροκινητήρων μηχανών (electric machine drives). Παρουσιάζει τρεις διαφορετικούς τρόπους αξιοποίησης FPGA. Αρχικά χρησιμοποιεί την πλατφόρμα PYNQ της Xilinx και τις βιβλιοθήκες Python. Η πλατφόρμα αυτή διαθέτει τρία επίπεδα (layers), upper, middle και lower layer. Το upper layer χρησιμοποιείτε κυρίως για την ανάπτυξη λογισμικού σε γλώσσα Python και την χρήση του Jupyter Notebook. Στο μεσαίο επίπεδο, το PYNQ περιλαμβάνει ένα λειτουργικό σύστημα που βασίζεται σε Linux, για το Jupyter Notebook και ένα σύνολο προγραμμάτων οδήγησης για αλληλεπίδραση με διάφορα στοιχεία του συστήματος υλικού Zynq. Έτσι, η προσπάθεια σχεδιασμού που απαιτείται για την ανάπτυξη κοινών στοιχείων λογισμικού ενός ενσωματωμένου συστήματος μειώνεται σημαντικά. Το κάτω στρώμα του PYNQ αντιπροσωπεύει ένα hardware system design, Στο PYNQ, τα σχέδια συστημάτων υλικού αναφέρονται συχνά ως επικαλύψεις (overlays) και μπορούν να χρησιμοποιηθούν με τρόπο παρόμοιο με τις βιβλιοθήκες λογισμικού. Συγκεκριμένα, το PYNQ παρέχει ένα βασικό σύστημα υλικού, το οποίο περιλαμβάνει σχεδόν όλα τα modules για επαναχρησιμοποίηση, όπως μπλοκ διασύνδεσης για DMA, ήχο, βίντεο, I2C και στοιχεία για logic tools. Οι επιταχυντές νευρωνικών δικτύων μπορούν στη συνέχεια να υλοποιηθούν μέσω αυτών των εργαλείων. Ο δεύτερος τρόπος παρουσιάζει την δυνατότητα που προσφέρει το MATLAB για την δημιουργία κώδικα VHDL για την αξιοποίηση του σε προγραμματισμό πλακετών της Xilinx, Microsemi και Intel. Με τον HDL Coder, ο προγραμματισμός FPGA για εφαρμογές ελέγχου κινητήρα που βασίζονται σε ML μπορεί να επιτευχθεί σε υψηλό επίπεδο και ο παραγόμενος κώδικας HDL μπορεί να εισαχθεί και να μεταγλωττιστεί χρησιμοποιώντας το Intel Quartus ή το Xilinx ή τέλος το Vivado Design Suite. Τέλος ο τρίτος τρόπος είναι μέσω του DPU IP

core της Xilinx, το οποίο είναι ένα high-level synthesis εργαλείο, που είναι ικανό να κάνει compile κώδικα C/C++ σε hardware. Το DPU διαθέτει ένα εξειδικευμένο σύνολο οδηγιών, το οποίο του επιτρέπει να λειτουργεί αποτελεσματικά σε πολλά CNN. Το συγκεκριμένο εργαλείο χρησιμοποιείται κυρίως για εφαρμογές μηχανικής όρασης και image recognition. [19]

Στο επόμενο άρθρο γίνεται προσέγγιση του τρόπου διαχείρισης της απαιτούμενης ενέργειας μιας πλακέτας FPGA με την χρήση μηχανικής μάθησης. Το άρθρο αυτό ονομάζεται “Shrinking FPGA Static Power via Machine Learning-Based Power Gating and Enhanced Routing” και οι συγγραφείς του είναι οι Zeinab Seifoori, Hossein Asadi και Mirjana Stojilović. Ο σκοπός του συγκεκριμένου άρθρου είναι να εντοπίσει τον πιο αποδοτικό αλγόριθμο για την καλύτερη διαχείριση της ενεργειακής κατανάλωση που έχει η πλακέτα FPGA. Σε αυτό το άρθρο, προτείνεται ένα σύνολο νέων αλγορίθμων για power-gating. Αυτοί οι αλγόριθμοι βασίζονται όλοι στην ομαδοποίηση K-means, μια πολύ γνωστή προσέγγιση ML που στοχεύει στην ομαδοποίηση αντικειμένων με βάση την ομοιότητά τους και μπορεί να προσαρμοστεί στο συγκεκριμένο πρόβλημα ομαδοποίησης. Στα πλαίσια αυτά σχεδιάστηκε ένας βελτιωμένος FPGA router, ο οποίος λαμβάνει υπόψη τη διαθεσιμότητα των περιοχών παροχής ενέργειας και βοηθά περαιτέρω την αποτελεσματική χρήση τους. Τα πειραματικά αποτελέσματα ήταν ότι όταν η ενεργοποίηση της πύλης ισχύος εφαρμόζεται μαζί με την ομαδοποίηση SiM-IPR-MP, παρατηρείται πρόσθετη μείωση ≈6 % στην κατανάλωση ισχύος, κατά μέσο όρο.

Στο τελευταίο άρθρο με τίτλο Application of Machine Learning in FPGA EDA Tool Development των Pingakshya Goswami και Dinesh Bhatia , παρουσιάζεται η εφαρμογή της μηχανικής μάθησης στον τομέα του EDA (Electronics Design Automation), δίνοντας έμφαση στα εργαλεία σχεδιασμού FPGA. Αναλύει πως η ML μπορεί να εφαρμοστεί στην κατανάλωση του FPGA, στο High Level Synthesis και Register Transfer Level αλλά και στο εργαλείο σχεδιασμού με την βοήθεια υπολογιστή (CAD). Το άρθρο αναλύει πως σχεδόν όλοι οι διαθέσιμοι αλγόριθμοι μηχανικής μάθησης είναι πλέον εφαρμόσιμοι σε EDA εργαλεία, μερικά από τα οπόια είναι τα AMD Xilinx Vitis AI και Synopsys DSO όπου χρησιμοποιούν ML ως μέρος της εφαρμογής των αλγορίθμων σε FPGA. Τέλος αναφέρει ότι οι αναλυτικές και ευρετικές μέθοδοι είναι πολύ αργές για μεγάλα σχέδια και για να μειωθεί ο χρόνος γινόταν ταυτόχρονη χρήση GPU και CPU. Επιπλέον αναλυτές κατάφεραν να επιταχύνουν τον χρόνο προσομοίωσης και ανάλυσης ακόμα περισσότερο κάνοντας χρήση της μεθόδου ενισχυμένης μάθησης (RL).

3. Εκπαίδευση Νευρωνικού δικτύου

3.1. Εισαγωγή

Σε αυτό το κομμάτι της εργασίας γίνεται ο σχεδιασμός, η υλοποίηση και η αξιολόγηση ενός Συνελικτικού Νευρωνικού Δικτύου (CNN) προσαρμοσμένου για χειρόγραφη αναγνώριση ψηφίων. Αξιοποιώντας την βιβλιοθήκη Keras για καθορισμό και εκπαίδευση μοντέλου, εφαρμογή που αναπτύχθηκε εμβαθύνει στην προεπεξεργασία, στον ορισμό της αρχιτεκτονικής του μοντέλου, στην εκπαίδευση, στην αξιολόγηση και, τέλος, στη μετατροπή για ανάπτυξη FPGA χρησιμοποιώντας το κιτ εργαλείων hls4ml.

Η χειρόγραφη αναγνώριση ψηφίων παραμένει ένα θεμελιώδες πρόβλημα στο επιστημονικό πεδίο της όρασης υπολογιστών, λειτουργώντας ως σημείο αναφοράς για νέες τεχνικές και τεχνολογίες. Η εφαρμογή που αναπτύχθηκε χρησιμοποιεί την τεχνική της διοχέτευσης (pipeline) - από την προεπεξεργασία δεδομένων έως τη μετατροπή μοντέλου για υλοποίηση σε FPGA.

Η μετατροπή αυτή πραγματοποιήθηκε με την βιβλιοθήκη hls4ml. Το συγκεκριμένο πακέτο έχει σχεδιαστεί για να μετατρέπει προ-εκπαιδευμένα μοντέλα μηχανικής μαθησης, σε ψηφιακά κυκλώματα. Αυτό έχει σαν αποτέλεσμα να αξιοποιείται η επεξεργαστική ισχύς που έχουν οι διατάξεις FPGA.

3.2. HLS4ML

Το hls4ml αποτελεί ένα εργαλείο high level synthesis με σκοπό την εφαρμογή αλγορίθμων μηχανικής μάθησης σε FPGA. Αυτό το κομμάτι της εργασίας παρέχει μια αναφορά στο πακέτο hls4ml, δίνοντας έμφαση στις δυνατότητές του και τις εφαρμογές του.

Το πλαίσιο hls4ml είναι ικανό να επεξεργάζεται ένα προ εκπαιδευμένο μοντέλο μηχανικής εκμάθησης (που προέρχεται από πλατφόρμες όπως το TensorFlow ή το Keras) και το μετατρέπει σε κώδικα γλώσσας περιγραφής υλικού.

Η σύνθεση υψηλού επιπέδου (High Level Synthesis – HLS) είναι μια προηγμένη διαδικασία κατά την οποία χρησιμοποιούνται γλώσσες προγραμματισμού υψηλού επιπέδου (κυρίως C ή C++) για τη δημιουργία σχεδίων υλικού. Η έξοδος ενός εργαλείου HLS μπορεί να χρησιμοποιηθεί για τη δημιουργία διαμόρφωσης ενός FPGA.

Σύμφωνα με τις πιο πρόσφατες ενημερώσεις, το πακέτο hls4ml υποστηρίζει κατά κύριο λόγο δομές προώθησης τροφοδοσίας, όπως πολυστρωματικά perceptrons (Multi-Layer Perceptron - MLP) και συνελικτικά νευρωνικά δίκτυα (CNN).

Η συγκεκριμένη βιβλιοθήκη ενσωματώνει εργαλεία που διευκολύνουν την ακρίβεια του μοντέλου και την απεικόνιση της απόδοσης, επιτρέποντας στους χρήστες να εκτελούν τεκμηριωμένες αποφάσεις σχετικά με την ισορροπία μεταξύ ακρίβειας και κατανάλωσης πόρων.

Αν και το hls4ml έχει τη δυνατότητα για ποικίλες εφαρμογές, αρχικά σχεδιάστηκε για χρήση στη Φυσική Υψηλών Ενεργειών και συγκεκριμένα στον Μεγάλο Επιταχυντή Αδρονίων (LHC) στην επεξεργασία μεγάλου όγκου δεδομένων σε πραγματικό χρόνο, αξιοποιώντας τη μηχανική μάθηση με επιτάχυνση και την χρήση διατάξεων FPGA.

Το πλαίσιο hls4ml αποτελεί απόδειξη της συγχώνευσης τεχνολογιών μηχανικής μάθησης και FPGA. Μεταφράζοντας μοντέλα ML σε διαμορφώσεις FPGA, δημιουργεί οδούς για συμπεράσματα ML που χαρακτηρίζονται από ελάχιστο λανθάνοντα χρόνο, αυξημένη απόδοση και ανώτερη ενεργειακή απόδοση. Όπως όλες οι μεθοδολογίες με επιτάχυνση υλικού, απαιτεί λεπτομερή κατανόηση και προσεκτικούς σχεδιασμούς, ωστόσο τα πιθανά πλεονεκτήματα σε επιλεγμένες εφαρμογές είναι αναμφισβήτητα σημαντικά.

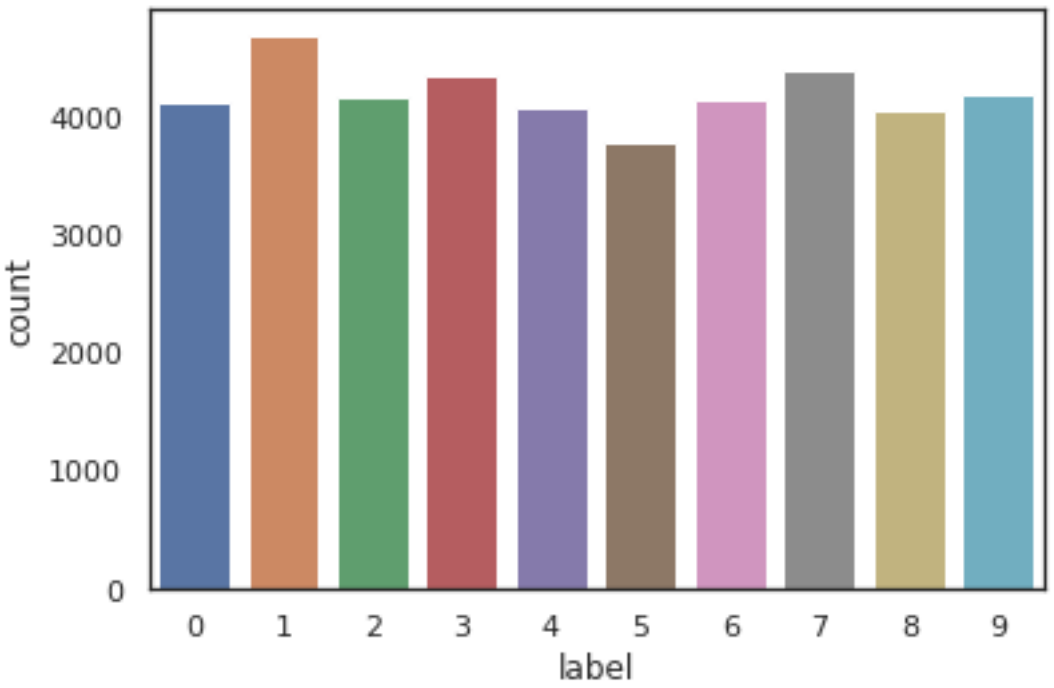
3.3. Ανάπτυξη Νευρωνικού

Τα δεδομένα φορτώνονται και προεπεξεργάζονται μέσω διαφόρων βημάτων ώστε να έχουν την κατάλληλη μορφή για την εφαρμογή του εργαλείου HLS4ML.

Επιγραμματικά τα βήματα είναι τα εξής:

- Εξαγωγή ετικετών και χαρακτηριστικών (labels and features)
- Κανονικοποίηση δεδομένων διασφαλίζοντας ότι οι τιμές των pixel βρίσκονται μεταξύ 0 και 1.
- Αναμόρφωση για να διασφαλιστεί η συμβατότητα με τα στρώματα Keras
- Ενιαία κωδικοποίηση ετικετών για τη διευκόλυνση της κατηγοριοποίησης κατά τη διάρκεια της εκπαίδευσης

Τα εργαλεία οπτικοποίησης, όπως το Seaborn, χρησιμοποιούνται για τη διερεύνηση της κατανομής των ετικετών στο σετ εκπαίδευσης, διασφαλίζοντας ένα ισορροπημένο σύνολο δεδομένων. Στην εικόνα 1 φαίνονται οι ετικέτες που εντοπίζονται στο σετ εικόνων που χρησιμοποιείται.



Εικόνα 1 Ετικέτες

Το CNN έχει σχεδιαστεί με τα ακόλουθα επίπεδα:

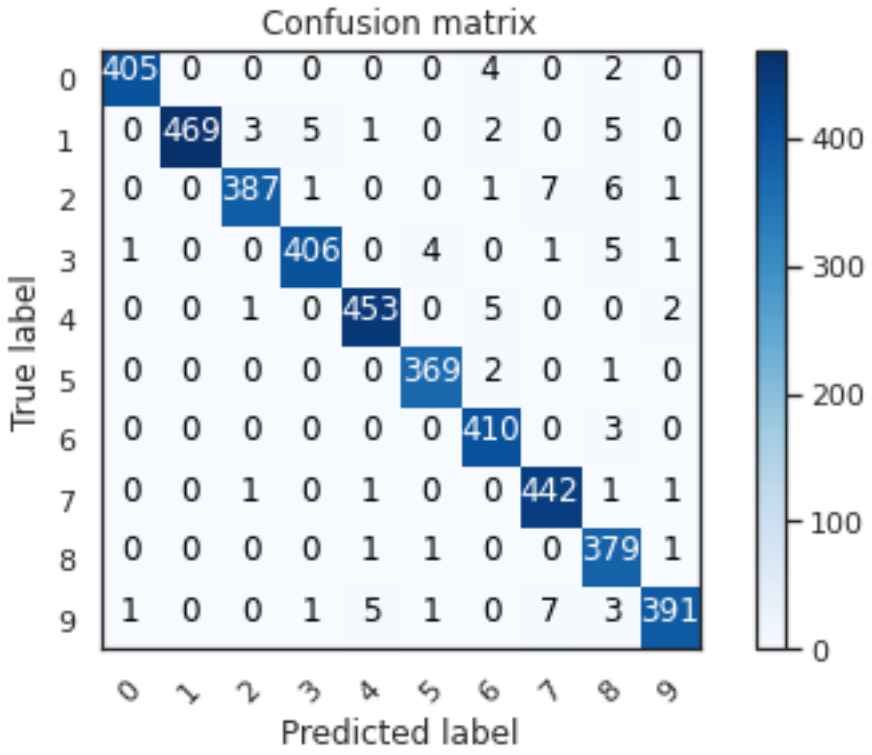
- Δύο επίπεδα Conv2D (32 φίλτρα, 5x5 πυρήνας) ακολουθούμενα από ένα επίπεδο MaxPooling και Dropout
- Δύο επίπεδα Conv2D (64 φίλτρα, πυρήνας 3x3) ακολουθούμενα από ένα άλλο επίπεδο MaxPooling και Dropout

- Flattening, ακολουθούμενη από πυκνή στρώση (256 μονάδες) και Dropout
- Ένα τελικό πυκνό στρώμα με 10 μονάδες (που αντιπροσωπεύουν 10 ψηφία) με ενεργοποίηση softmax

Αυτός ο σχεδιασμός στοχεύει στην εξαγωγή των χαρακτηριστικών από τις εικόνες, που κυμαίνονται από βασικές ακμές έως πιο σύνθετα μοτίβα.

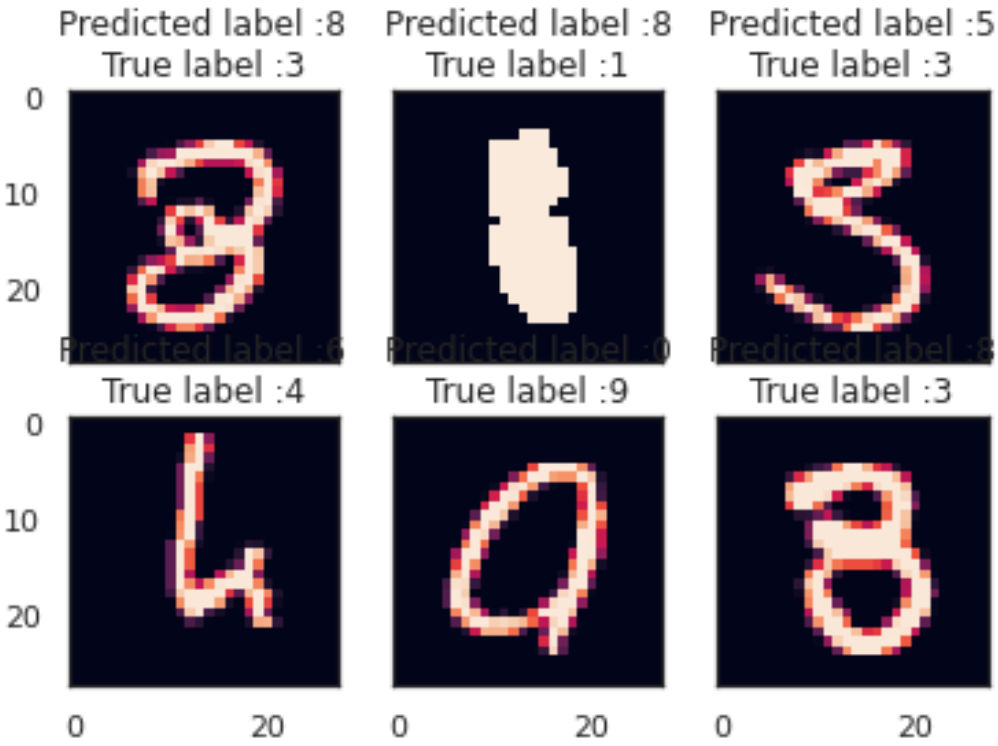
Χρησιμοποιείται ένας βελτιστοποιητής (optimizer) RMSprop με κατηγορηματική cross-entropy loss. Ο συγκεκριμένος όρος χρησιμοποιείται σε συστήματα μηχανικής μάθησης και αναφέρεται σε μια συνάρτηση απώλειας (loss). Ουσιαστικά υπολογίζει την διαφορά μεταξύ των πιθανοτήτων της σωστής κατανομής των labels και της προβλεπόμενης κατανομής που δημιουργείται από το μοντέλο της μηχανικής μάθησης. Για την καταπολέμηση της πιθανής υπερπροσαρμογής (overfitting) και την ενίσχυση της γενίκευσης, ενσωματώνονται τεχνικές αύξησης δεδομένων, όπως περιστροφές, σμικρύνσεις και μετατοπίσεις. Ένας learning rate annealer εφαρμόζεται επίσης για να μειώνει προσαρμοστικά τον ρυθμό εκμάθησης όταν η ακρίβεια είναι σταθερή.

Η απόδοση του εκπαιδευμένου μοντέλου αξιολογείται χρησιμοποιώντας δεδομένα επικύρωσης (validation data), καμπύλες ακρίβειας γραφικής παράστασης και απώλειας (accuracy and loss plots) τόσο για σύνολα δεδομένων εκπαίδευσης όσο και για σύνολα δεδομένων επικύρωσης (validation data). Ένας πίνακας σύγχυσης ή αλλιώς confusion matrix , εικόνα 2, υπογραμμίζει περιοχές όπου το μοντέλο μπορεί να συγχέει ένα ψηφίο με ένα άλλο, παρέχοντας πληροφορίες για πιθανούς τομείς βελτίωσης.



Εικόνα 2 Confusion Matrix

Αφού ολοκληρωθεί η εκπαίδευση του μοντέλου εφαρμόστηκε στις εικόνες test για να οπτικοποιηθούν τα αποτελέσματα. Η παρακάτω εικόνα 3 παρουσιάζει ορισμένα λανθασμένα αποτελέσματα εντοπισμού του σωστού αριθμού.



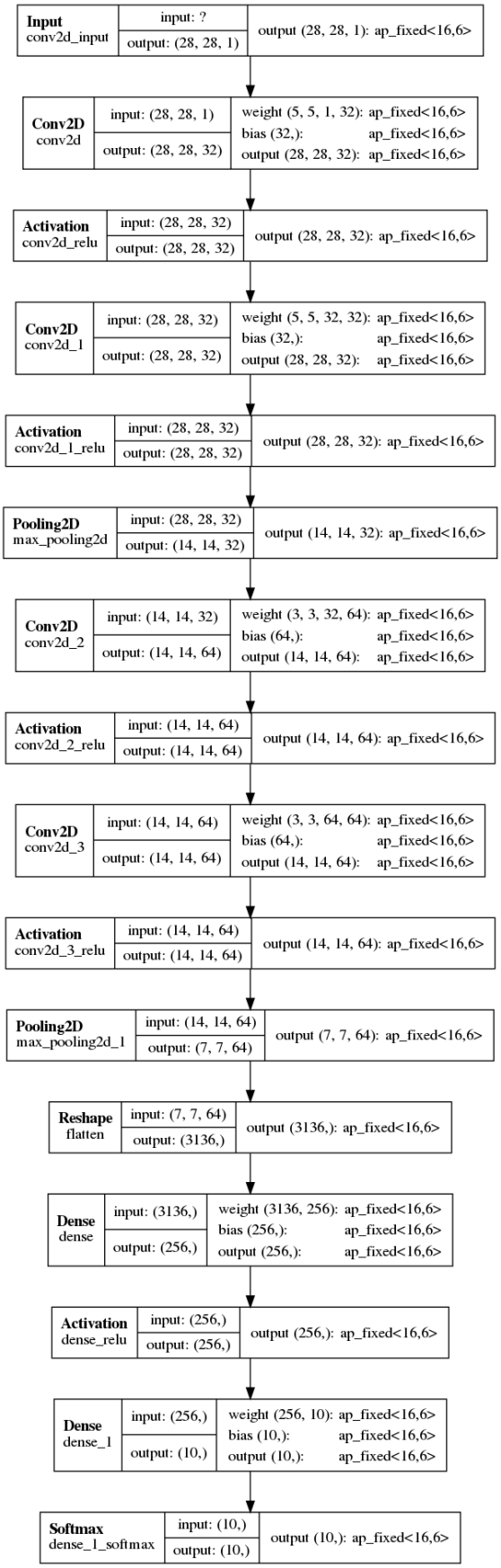
Εικόνα 3 Αποτελέσματα

Το τελικό μοντέλο, ενώ έχει δημιουργηθεί σε ένα τυπικό υπολογιστικό περιβάλλον python, μετατρέπεται επίσης σε ένα πακέτο που μπορεί να χρησιμοποιηθεί για την ανάπτυξη του ίδιου μοντέλου σε FPGA (Field-Programmable Gate Array) χρησιμοποιώντας την εργαλειοθήκη hls4ml. Το σκεπτικό πίσω από αυτό είναι η αξιοποίηση των δυνατοτήτων παράλληλης επεξεργασίας των FPGA, ιδιαίτερα επωφελών για τα CNN.

Το σενάριο απεικονίζει τη δύναμη και την ευελιξία των σύγχρονων βιβλιοθηκών και εργαλείων μηχανικής εκμάθησης. Ξεκινώντας από τα αρχικά δεδομένα και καταλήγοντας σε ένα μοντέλο έτοιμο για FPGA, κάθε βήμα δείχνει πώς τα Python, Keras και hls4ml μπορούν να ενσωματωθούν απρόσκοπτα για να αντιμετωπίσουν τις πραγματικές προκλήσεις όρασης υπολογιστών.

Ο κώδικας λοιπόν που οδηγεί σε αυτό το απότελεσμα ξεκινάει με την εισαγωγή ‘import’ κάποιον βασικών βιβλιοθηκών Python που χρησιμοποιούνται για ανάλυση και επεξεργασία δεδομένων (pandas, numpy), καθώς και για την οπτικοποίηση δεδομένων (matplotlib, seaborn). Το %matplotlib inline ρυθμίζει το περιβάλλον του Jupyter notebook ώστε να εμφανίζει γραφήματα απευθείας. Τα δεδομένα εκπαίδευσης και δοκιμής διαβάζονται από αρχεία CSV και είναι εικόνες χειρόγραφων αριθμών. Οι ετικέτες (labels) απομονώνονται και τα χαρακτηριστικά (features) αποθηκεύονται ξεχωριστά. Η χρήση της seaborn για την καταμέτρηση των ετικετών παρέχει μια οπτική

εκτίμηση της κατανομής τους (εικόνα 1). Στη συνέχεια ολοκληρώνεται η κανονικοποίηση των δεδομένων όπου τα pixel των εικόνων πρέπει να βρίσκονται σε μια κλίμακα από 0 έως 1 για βέλτιστη εκπαίδευση του νευρωνικού δικτύου. Αυτό επιτυγχάνεται με την διαίρεση των δεδομένων με τον αριθμό 255, όπου είναι η μέγιστη τιμή που μπορεί να έχει ένα pixel, 'X_train = X_train / 255.0' και 'test = test / 255.0'. Ο ανασχηματισμός των δεδομένων στη σωστή διάσταση είναι απαραίτητη ώστε να ταιριάζουν με την είσοδο του νευρωνικού δικτύου (CNN). Επιπλέον, η κωδικοποίηση one-hot, η αναπαράσταση δηλαδή των αριθμών στο δυαδικό σύστημα, μετατρέπει τις αριθμητικές ετικέτες σε δυαδικό διάνυσμα, προκειμένου το δίκτυο να μπορεί να εκτιμήσει την πιθανότητα κάθε κατηγορίας ανεξάρτητα. Το νευρωνικό δίκτυο ορίζεται στοιχείο προς στοιχείο, από τα βασικά στοιχεία όπως τα στρώματα συνέλιξης (Conv2D) που ανακαλύπτουν τοπικά χαρακτηριστικά στις εικόνες, μέχρι στοιχεία συγκέντρωσης (MaxPooling) που μειώνουν τις διαστάσεις των χαρακτηριστικών, και στοιχεία τυχαίας απόρριψης (Dropout) που βοηθούν στην αποφυγή υπερεκπαίδευσης του δικτύου. Η συγκεκριμένη διαδικασία πραγματοποιείται με την χρήση της εντολής model.add όπου προστίθενται τα κατάλληλα στρώματα ώστε να δημιουργηθεί το δίκτυο. Για την ολοκλήρωση του καλείται η εντολή model.compile. Η τελική μορφή του νευρωνικού δικτύου φαίνεται στην εικόνα 4. Έχοντας δημιουργήσει το δίκτυο πρέπει να βελτιστοποιηθεί και να εκπαιδευτεί. Επιλέχθηκε ο RMSprop optimizer όπως αναλύθηκε παραπάνω, για την καλύτερη εκπαίδευση του νευρωνικού. Αφού ολοκληρωθεί η δημιουργία του νευρωνικού ακολουθεί η διαδικασία της επαύξησης των δεδομένων. Η συγκεκριμένη διαδικασία επεξεργάζεται το σύνολο δεδομένων, το σύνολο των εικόνων στην προκειμένη περίπτωση, για να δημιουργήσει νέο σύνολο με εικόνες διαφοροποιημένες από τις αρχικές ώστε το δίκτυο να εκπαιδευτεί. Η επεξεργασία που πραγματοποιείται είναι τυχαία, και αποτελείται από μετατοπίσεις των εικόνων είτε δεξιά είτε αριστερά (shift), περιστροφή των εικόνων (rotate), ορισμός του μέσου (mean) των δεδομένων στο 0 και διαίρεση αυτών με τον μέσο όρο τους (standard deviation) και τέλος το «γύρισμα» των εικόνων (flip) είτε οριζόντια είτε κάθετα. Αυτό το νέο δείγμα των εικόνων θα είναι το σύνολο των δεδομένων με το οποίο θα εκπαιδευτεί το νευρωνικό και ο λόγος των αλλαγών αυτών είναι για να να αναγνωρίζει και εικόνες οι οποίες δεν αποδίδουν ξεκάθαρα των αριθμό που απεικονίζουν. Το νευρωνικό θα είναι σε θέση έτσι να αναγνωρίζει τα χειρόγραφα νούμερα ακόμα και αν η υπό δοκιμασία εικόνα δεν είναι τέλεια αλλά έχει κάποια μορφής παραμόρφωση.



Εικόνα 4 Νευρωνικό Δίκτυο

4. Αναγνώριση χρώματος με χρήση ML

4.1. Εισαγωγή

Η ανίχνευση οδικών σημάτων σε εικόνες είναι μια κρίσιμη εργασία σε διάφορες εφαρμογές όπως η αυτόνομη οδήγηση, η πλοήγηση επαυξημένης πραγματικότητας και η επιτήρηση της κυκλοφορίας. Με την έλευση της βαθιάς μάθησης και την εκθετική αύξηση της υπολογιστικής ισχύος, η εκπαίδευση νευρωνικών δικτύων (NN) με τη χρήση FPGA έχει γίνει αρκετά διαδεδομένη λόγω της ανώτερης απόδοσής τους σε σχέση με τις παραδοσιακές τεχνικές όρασης όπου γίνεται χρήση συμβατικών επεξεργαστών (CPU) και καρτών γραφικών (GPU).

Ορισμός προβλήματος για την αναγνώριση των σημάτων οδικής κυκλοφορίας.

Δίνεται μια εικόνα εισόδου I. Στόχος είναι να εντοπιστούν όλες οι περιοχές $R = \{r_1, r_2, \dots, r_n\}$ στην εικόνα I όπου υπάρχουν οδικές πινακίδες και να ταξινομηθεί η κάθε περιοχή σε μια συγκεκριμένη κατηγορία οδικών πινακίδων $C = \{c_1, c_2, \dots, c_m\}$.

Η έξοδος είναι επομένως ένα σύνολο οριοθετημένων πλαισίων μαζί με τις σχετικές ετικέτες κατηγορίας.

Το επόμενο βήμα είναι η προετοιμασία δεδομένων. Συλλέγονται εικόνες που περιέχουν οδικές πινακίδες από διάφορα περιβάλλοντα, συνθήκες φωτισμού και γωνίες. Ένα ποικίλο σύνολο δεδομένων εξασφαλίζει στιβαρότητα σε εφαρμογές πραγματικού κόσμου. Για να ενισχυθεί η ποικιλομορφία του συνόλου εκπαίδευσης και να αποφευχθεί η υπερβολική προσαρμογή, γίνεται χρήση διάφορων μετασχηματισμών, όπως περιστροφή, κλιμάκωση, περικοπή, ρύθμιση φωτεινότητας και οριζόντια ανατροπή (horizontal flip), περιστροφή δηλαδή μιας εικόνας ή μιας περιοχής της εικόνας κατά γωνία προς τα αριστερά ή προς τα δεξιά, ώστε να δημιουργηθεί μια οριζόντια αναστροφή ή κλίση, στις αρχικές εικόνες.

Τα Νευρωνικά δίκτυα (NN) δεδομένης της ικανότητάς τους να μαθαίνουν ιεραρχικά μοτίβα στις εικόνες αποτελούν τη βασική αρχιτεκτονική για το σύστημα ανίχνευσης οδικών σημάτων επιτρέποντας στο NN να εντοπίσει διαφορετικές οδικές πινακίδες.

Η εκπαίδευση ενός νευρωνικού δικτύου για την ανίχνευση οδικών σημάτων σε εικόνες συνδυάζει αρχιτεκτονικές αποφάσεις, διαδικασίες αύξησης δεδομένων και τεχνικές βελτιστοποίησης για να δημιουργήσει ένα ισχυρό μοντέλο ικανό για ακριβή εντοπισμό και ταξινόμηση. Με τα σωστά εργαλεία και διαδικασίες, τα NN προσφέρουν μια πολλά υποσχόμενη λύση στην πρόκληση του εντοπισμού οδικών πινακίδων σε διάφορα σενάρια πραγματικού κόσμου.

4.2.Νευρωνικό Δίκτυο Εκπαίδευση και εφαρμογή

Η εκπαίδευση του NN πραγματοποιήθηκε με την χρήση του Octave και δύο script που αναλύονται παρακάτω.

Ο κύριος στόχος του πρώτου Script είναι η υλοποίηση και η εκπαίδευση ενός μοντέλου νευρωνικών δικτύων, που έχει ως αποστολή την κατηγοριοποίηση μεμονωμένων pixel από μια εικόνα σε δύο διακριτές ταξινομήσεις. Αυτό επιτυγχάνεται με βάση τις εγγενείς τιμές RGB των εικονοστοιχείων και των παρεχόμενων ετικετών.

Το NN εκπαιδεύεται με την χρήση του αλγορίθμου Gradient descent ο οποίος είναι ένας επαναληπτικός αλγόριθμος βελτιστοποίησης πρώτης τάξης που χρησιμοποιείται για να βρει το ελάχιστο μιας συνάρτησης. Στο πλαίσιο της μηχανικής μάθησης και της βαθιάς μάθησης, ο αλγόριθμος αυτός χρησιμοποιείται για την ελαχιστοποίηση μιας συνάρτησης κόστους, ρυθμίζοντας έτσι τις παραμέτρους του μοντέλου μας.

Ακολουθεί μια βασική επισκόπηση του αλγόριθμου gradient descent για την εκπαίδευση ενός μοντέλου:

1. Αρχικοποίηση ενός συνόλου παραμέτρων.

2. Υπολογισμός της κλίσης της συνάρτησης κόστους σε σχέση με κάθε παράμετρο. Η κλίση είναι ένα διάνυσμα που δείχνει προς την κατεύθυνση της πιο απότομης αύξησης της συνάρτησης.

3. Ενημέρωση παραμέτρων αντίθετα της κατεύθυνσης κλήσης:

$$\theta_{new} = \theta_{old} - a\ x \nabla J(\theta)$$

Όπου:

- θ είναι το διάνυσμα παραμέτρου.

- α είναι ο ρυθμός εκμάθησης, μια παράμετρος που καθορίζει το μέγεθος του βήματος σε κάθε επανάληψη ενώ κινείται προς ένα ελάχιστο της συνάρτησης κόστους.

- $\nabla J(\theta)$ είναι η κλίση της συνάρτησης κόστους.

4. Δοκιμή σύγκλισης βασιζόμενη στο γεγονός εάν η διαφορά μεταξύ του παλιού κόστους και του νέου κόστους είναι κάτω από ένα συγκεκριμένο όριο ή εάν έχει επιτευχθεί ένας προκαθορισμένος αριθμός επαναλήψεων. Εάν όχι, επιστροφή στο βήμα 2.

5. Τερματισμός μόλις ικανοποιηθούν τα κριτήρια σύγκλισης ή επιτευχθεί ένας προκαθορισμένος αριθμός επαναλήψεων. [20][21]

Έχοντας αναφερθεί στον αλγόριθμο που χρησιμοποιήθηκε για την εκπαίδευση του NN σειρά έχει η ουσιαστική εκπαίδευση.

Στη συνέχεια ορίστηκε σχολαστικά μια σειρά από θεμελιώδεις παραμέτρους. Αυτά περιλαμβάνουν τον προκαθορισμένο αριθμό των training epoch, το specified learning rate για τις ρυθμίσεις βάρους του μοντέλου και τις ακριβείς διαστάσεις των εικόνων που εξετάζουμε. Σημαντικό είναι να τονιστεί και η επιλογή ενός τυχαίου σπόρου (random seed), ο οποίος εγγυάται τη δυνατότητα αναπαραγωγής των αποτελεσμάτων.

Βασικό βήμα στην σωστή λειτουργία της διαδικασίας είναι οι δύο κεντρικές εικόνες, την κύρια εικόνα εκπαίδευσης (Εικόνα 5) και τη συμπληρωματική εικόνα (Εικόνα 6) που περιέχει τις ετικέτες. Αυτές οι εικόνες χρησιμεύουν ως πηγή δεδομένων και ως σημείο αναφοράς, αντίστοιχα.

Αυτές οι εικόνες υποβάλλονται σε μια σειρά βημάτων προ επεξεργασίας. Αυτό περιλαμβάνει όχι μόνο μετατροπές τύπων δεδομένων, αλλά και την αναμόρφωση και την αναδιάρθρωση των δεδομένων σε μια μορφή που είναι κατάλληλη για εκπαίδευση νευρωνικών δικτύων. Αρχικά οι τιμές των εικονοστοιχείων τους μετατρέπονται σε δεκαδικούς αριθμούς διπλής ακρίβειας και αυτό γίνεται με την χρήση της εντολής: inputPicture = cast(inputPicture,'double');

Στη συνέχεια μετατρέπει τις εικόνες από τρισδιάστατο πίνακα (R,G,B) σε δύο διαστάσεων: inputPicture = reshape(inputPicture,[],3); Τέλος κανονικοποιεί την εικόνα διαιρώντας τα εικονοστοιχεία με το 255, που είναι η μέγιστη τιμή ενός 8-bit καναλιού χρώματος, μετατρέποντας έτσι τις τιμές σε ένα εύρος από 0 έως 1: inputPicture = inputPicture/255;

Το σενάριο υιοθετεί μια παραδοσιακή δομή νευρωνικών δικτύων. Αυτή η αρχιτεκτονική διαιρείται σε ένα στρώμα εισόδου που περιλαμβάνει τρεις νευρώνες (που αντικατοπτρίζουν απευθείας τα κανάλια RGB των εικόνων), και ένα τελικό στρώμα εξόδου με έναν νευρώνα.

Το νευρωνικό δίκτυο υποβάλλεται σε αυστηρή εκπαίδευση χρησιμοποιώντας τα δεδομένα της εικόνας. Σε όλο αυτό το επαναληπτικό πρόγραμμα εκπαίδευσης, τα εσωτερικά βάρη του δικτύου προσαρμόζονται περιοδικά, επηρεαζόμενα από τον προκαθορισμένο ρυθμό μάθησης και τον αριθμό των epoch. Αξιοσημείωτη είναι η καταγραφή του κόστους (ή της απώλειας) που παρατηρείται σε κάθε epoch, χρησιμεύοντας ως αξιολογική μέτρηση σε πραγματικό χρόνο της εξελισσόμενης απόδοσης του μοντέλου και της πορείας του. [22]

Με την ολοκλήρωση της εκπαίδευσης, το σενάριο μεταβαίνει στο στάδιο της

αξιολόγησης. Εδώ, το πρόσφατα εκπαιδευμένο μοντέλο αναπτύσσεται στην αρχική εικόνα εισόδου για τη δημιουργία προβλέψεων. Κάθε pixel, με βάση τις τιμές RGB που έχει, ταξινομείται σε μία από τις δύο προκαθορισμένες κατηγορίες, εάν περιέχει οδικό σήμα ή όχι. Αν το pixel περιέχει ένα οδικό σήμα τότε παίρνει την τιμή 1 και μεταφράζεται σε χρώμα άσπρο ενώ σε αντίθετη περίπτωση παίρνει την τιμή 0 δηλαδή μαύρο. Αυτές οι ταξινομήσεις οπτικοποιούνται, καθορίζουν δηλαδή τις νέες τιμές των pixel αποτυπώνοντας την τελική εικόνα, προσφέροντας στους ερευνητές και τους χρήστες μια οπτική αξιολόγηση της αποτελεσματικότητας και της ακρίβειας του μοντέλου σε σενάρια πραγματικού κόσμου.

Ως τελικό βήμα, το σενάριο ξεκινά το έργο της εξαγωγής, εμφάνισης και αρχειοθέτησης των τελικών εκπαιδευμένων βαρών του νευρωνικού δικτύου. Αυτά τα βάρη, στην ουσία, ενσωματώνουν τη «μαθημένη γνώση» του δικτύου, αντιπροσωπεύοντας τον βασικό του πίνακα λήψης αποφάσεων. Η εξαγωγή τους είναι ζωτικής σημασίας όχι μόνο για τον έλεγχο της απόδοσης αλλά και για την αναπαραγωγή της συμπεριφοράς του δικτύου σε διαφορετικά συστήματα, πλατφόρμες ή εφαρμογές πραγματικού κόσμου. Στη συνέχεια, το σενάριο προχωρά στην αποθήκευση τόσο της διαμόρφωσης του εκπαιδευμένου δικτύου όσο και των προβλέψεων που προκύπτουν, αποθηκεύοντας τες για μελλοντική χρήση ή περαιτέρω ανάλυση.

Συνοψίζοντας, αυτό το σενάριο προσφέρει ένα ισχυρό, ολοκληρωμένο σχέδιο για τη σχεδίαση, την εκπαίδευση και την αξιολόγηση ενός απλοϊκού νευρωνικού δικτύου σε εργασίες ταξινόμησης που βασίζονται στις τιμές του εκάστοτε pixel της εικόνας. Επιπλέον παραθέτει έναν πίνακα με τα βάρη που προκύπτουν τόσο για τους νευρώνες εισόδου όσο και για τον νευρώνα εξόδου. Τα βάρη αυτά είναι σε μορφή κινητής υποδιαστολής. Το επόμενο Script έχει στόχο τη μετατροπή των παραμέτρων αυτών σε αναπαράσταση αριθμών σταθερής υποδιαστολής (fixed-point arithmetic). Τέτοιες μετατροπές είναι συνήθως αναγκαίες όταν γίνεται χρήση ενσωματωμένων συστημάτων διότι δίνουν την δυνατότητα μιας καλύτερης υπολογιστικής απόδοσης δίνοντας έμφαση στην καλύτερη απόδοση και όχι στην ακρίβεια των υπολογισμών

Κεντρικό στοιχείο στις λειτουργίες του δεύτερου σεναρίου είναι να φορτώνει ένα συγκεκριμένο αρχείο Octave με τίτλο 'NN_RGB_2_Categories_config.mat'. Αυτό το αρχείο φιλοξενεί τις προηγουμένως εκπαιδευμένες παραμέτρους (ή βάρη) του νευρωνικού δικτύου σε μορφή κινητής υποδιαστολής από το πρώτο script. Αυτές οι παράμετροι ουσιαστικά ενσωματώνουν τη «μαθημένη νοημοσύνη» του νευρωνικού δικτύου, καθιστώντας την ακριβή μετατροπή και αναπαράστασή τους σε μορφή

σταθερής υποδιαστολής (fixed-point arithmetic) εξαιρετικά σημαντική για την συνέχιση της διαδικασίας.

Δύο βασικές μεταβλητές, factor και upscale, ορίζονται. Αυτές οι παράμετροι αξιοποιούνται για την μετάβασης από αναπαραστάσεις κινητής υποδιαστολής σε σταθερού σημείου των τιμών που έχουν τα βάρη. Το inputFactor προέρχεται από την παράμετρο upscale και καθορίζει την ευαισθησία της αναπαράστασης για τις τιμές εισόδου στη μορφή σταθερού σημείου.

Η διαδικασία μετατροπής χωρίζεται σε τρία βήματα, την μετατροπή των βαρών, τη μετατροπή του bias και την μετατροπή των τύπων δεδομένων.

Μετασχηματισμός βαρών δικτύου: Οι παράμετροι (ή βάρη) του νευρωνικού δικτύου υφίστανται συστηματικούς μετασχηματισμούς με βάση τον καθορισμένο παράγοντα factor. Αυτός ο μετασχηματισμός κλιμακώνει τα βάρη, καθιστώντας τα κατάλληλα για μετατροπή σε ακέραιους αριθμούς.

Μετασχηματισμός bias: Στα Νευρωνικά δίκτυα, η μονάδα bias είναι αναπόσπαστο κομμάτι της μετατόπισης της συνάρτησης ενεργοποίησης και είναι η τέταρτη στήλη στους πίνακες βάρους. Το σενάριο διασφαλίζει ότι αυτές οι τιμές προκατάληψης bias υφίστανται έναν πρόσθετο μετασχηματισμό που επηρεάζεται από την παράμετρο αναβάθμισης upscale.

Μετατροπή τύπου δεδομένων: Μετά την κλιμάκωση, το script αναλαμβάνει το κρίσιμο βήμα της μετατροπής τύπου δεδομένων. Οι αναπαραστάσεις κινητής υποδιαστολής των παραμέτρων του νευρωνικού δικτύου μετατρέπονται σε μορφή ακέραιου αριθμού 32 bit (ή 'int32'), πραγματοποιώντας έτσι τη μετάβασή τους σε μορφή σταθερού σημείου.

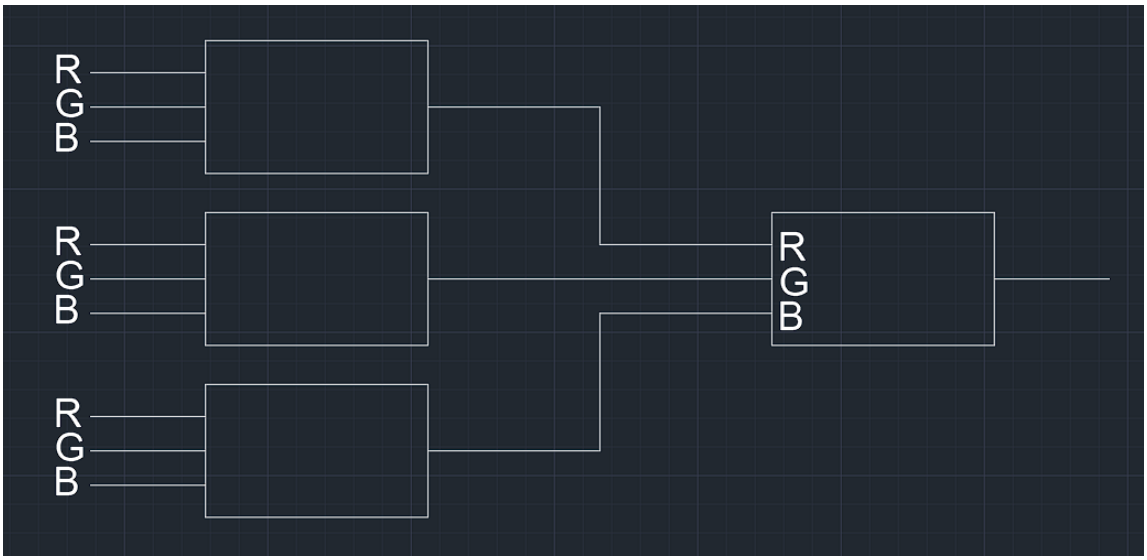
Η μετατροπή αυτή πραγματοποιείται με τις εξής γραμμές κώδικα.

Πρώτα με την χρήσης της δύναμης του 2 (2^factor) μετατοπίζεται το δεκαδικό μέρος των αριθμών ενώ η γραμμή του κώδικα που επιτυγχάνει αυτή την μετατόπιση είναι η: network1 = ((2^factor)) * nnParams{1};

Στη συνέχεια ακολουθείται η ίδια διαδικασία για το bias των νευρώνων με την χρήση της μεταβλητής upscale network1(:,4)=network1(:,4)*((2^upscale));

Τέλος εξάγει τις αναπαραστάσεις σταθερού σημείου των πινάκων βαρών του νευρωνικού δικτύου τόσο για το κρυφό επίπεδο όσο και για το επίπεδο εξόδου.

Το μοντέλο του νευρωνικού που δημιουργείται έχει την μορφή της εικόνας 5



Εικόνα 5Νευρωνικό Δίκτυο

Κάθε είσοδος R,G,B πολλαπλασιάζεται με το βάρος του νευρώνα και προστίθεται η τιμή του bias. Το αποτέλεσμα αυτής της πράξης εισάγεται εκ νέου στο επίπεδο εξόδου όπου πραγματοποιείται η ίδια πράξη με την διαφορά ότι αλλάζουν τα βάρη και το bias.

Για την επίτευξη του καλύτερου αποτελέσματος η εκπαίδευση πραγματοποιήθηκε με διάφορες εικόνες και τα εξαγόμενα δεδομένα παρουσιάζονται παρακάτω. Κάθε εκπαίδευση ξεκινά με μηδενικές τιμές στα βάρη του νευρώνα και δεν επηρεάζεται από την προηγούμενη εκπαίδευση.

Εικόνα 6 Αρχική εικόνα

Εικόνα 7 Εικόνα labels

Τα αποτελέσματα ήταν τα εξής :

Weight Matrix from the Input to the Hidden Layer

0.121881	0.659951	0.575945	0.456726
0.070677	1.749182	1.175333	-0.158142
0.422073	1.347577	1.052676	0.246278

Weight Matrix from the Hidden to the Output Layer

-0.2985	-1.0951	-1.1637	-0.7958

Αντίστοιχα η μετατροπή σε ακεραίους αριθμούς έδωσε τα εξής αποτελέσματα.

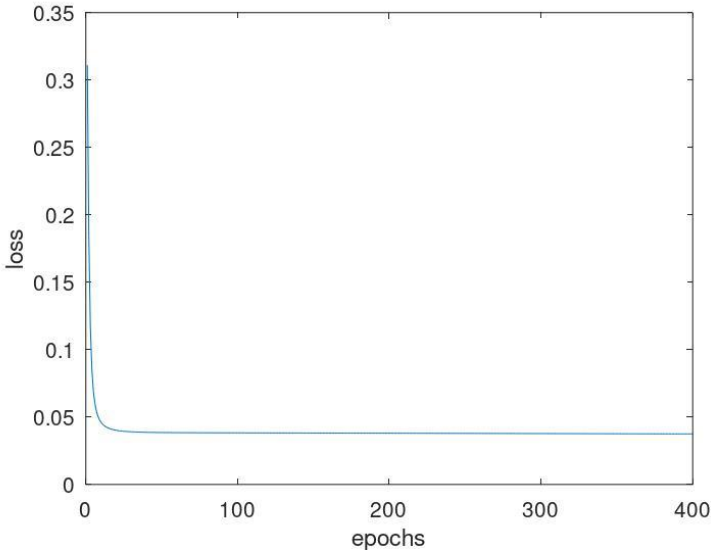
Fixed Point Matrix for Hidden Layer

4	21	18	3742
2	56	38	-1295
14	43	34	2018

Fixed Point Matrix for Output Layer

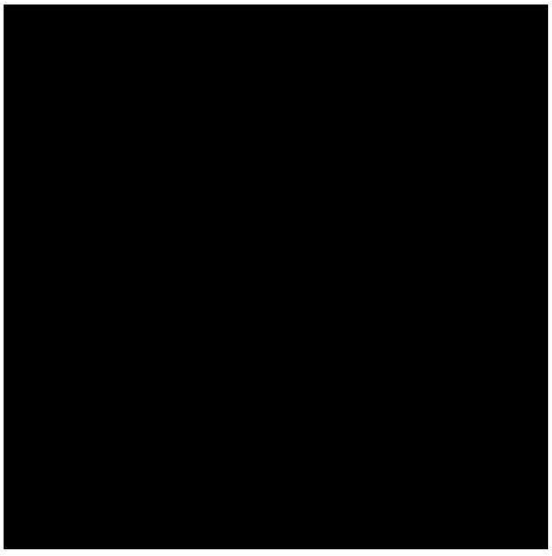
-10	-35	-37	-6519

Η γραφική παράσταση του κόστους για την συγκεκριμένη εκμάθηση είναι η παρακάτω εικόνα.



Εικόνα 8 Loss/Epochs

Τέλος το αποτέλεσμα με το εκπαιδευμένο NN όταν εφαρμοστεί εκ νέου στην εικόνα ήταν η εικόνα 9.



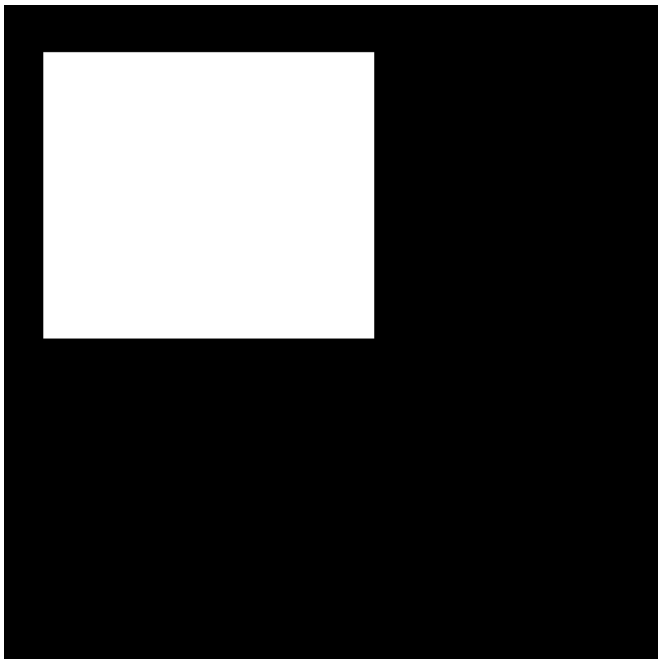
Εικόνα 9 Αποτελέσματα Νευρωνικού

Είναι εύκολα αντιληπτό ότι η εικόνα του αποτελέσματος δεν είναι σωστή. Το NN δεν κατάφερε να εντοπίσει σωστά την πινακίδα με αποτέλεσμα να ‘μαυρίσει’ όλη την εικόνα.

Επομένως έγιναν άλλες δύο προσπάθειες εκπαίδευσης του νευρωνικού ώστε να βρεθούν τα βάρη με τα καλύτερα αποτελέσματα.

Με την χρήση της επόμενης εικόνας (Εικόνα 10,11) τα αποτελέσματα ήταν εμφανώς καλύτερα τόσο στην τελική εφαρμογή του NN στην εικόνα (Εικόνα 13) όσο και με την γραφική παράσταση του LOSS.

Εικόνα 10 Αρχική εικόνα



Εικόνα 11 Εικόνα labels

Weight Matrix from the Input to the Hidden Layer

-3.5044	1.1749	-1.4451	0.5360
4.1582	-0.6413	1.7890	-1.0677
5.9786	-1.6722	2.2692	-1.4530

Weight Matrix from the Hidden to the Output Layer

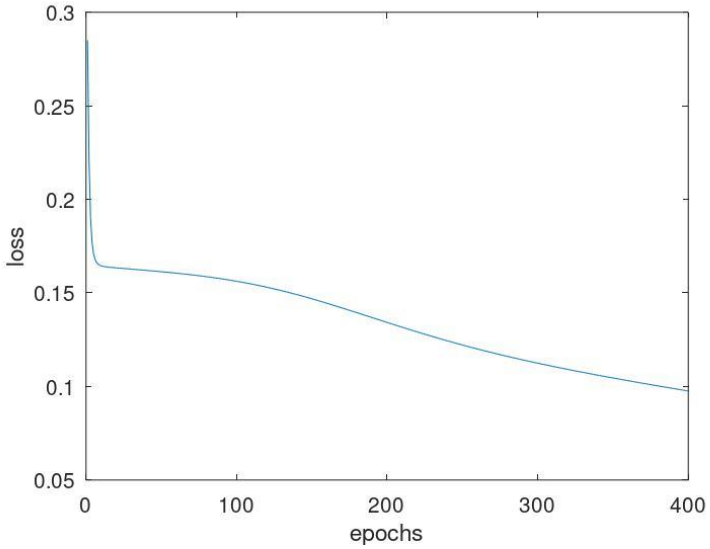
2.0885	-1.6366	-2.7008	1.2690

Fixed Point Matrix for Hidden Layer

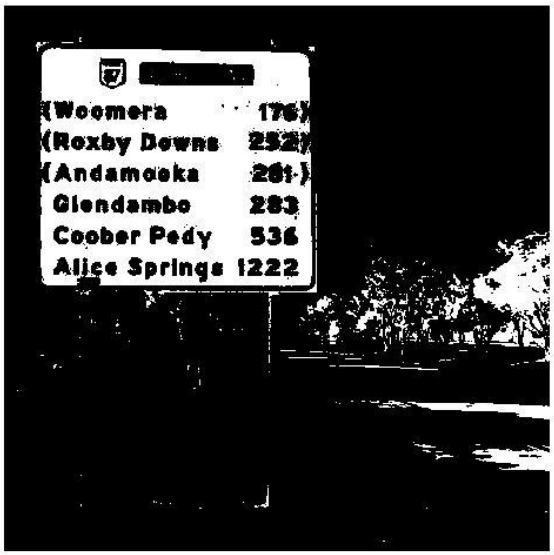
-112	38	-46	4391
133	-21	57	-8746
191	-54	73	-11903

Fixed Point Matrix for Output Layer

67	-52	-86	10396



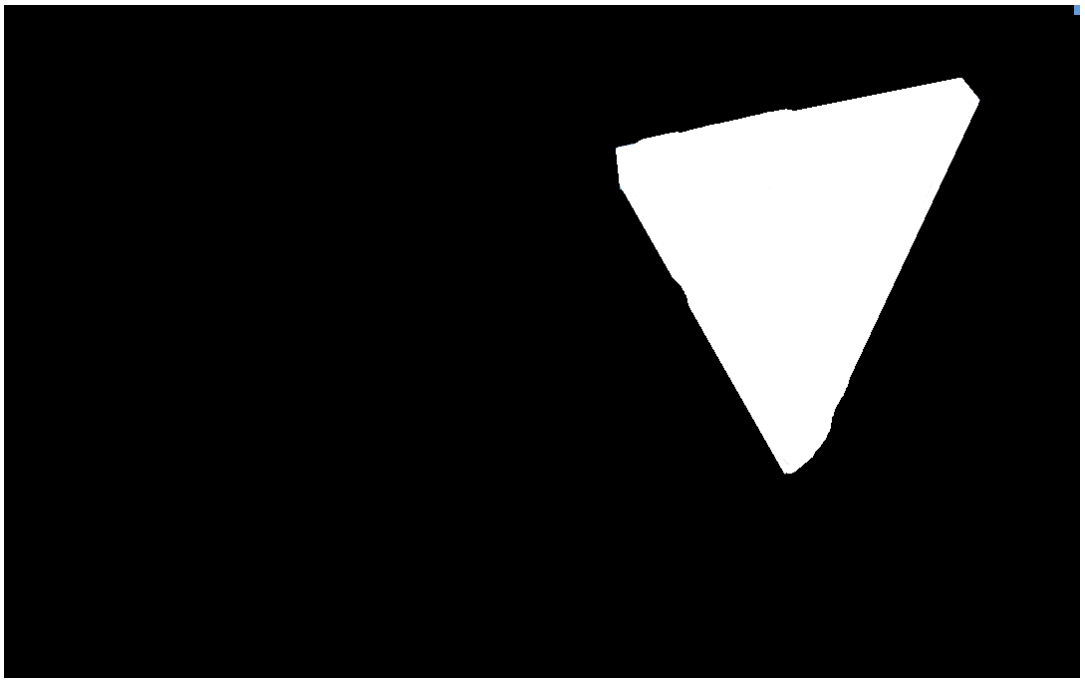
Εικόνα 12 Loss/Epochs



Εικόνα 13 Αποτελέσματα Νευρωνικού

Η τελευταία εκπαίδευση του Νευρωνικού είχε και τα καλύτερα αποτελέσματα όσον αφορά την εφαρμογή στην εικόνα 14.

Εικόνα 14 Αρχική εικόνα



Εικόνα 15 Εικόνα labels

Weight Matrix from the Input to the Hidden Layer

0.90314	-1.41457	-2.71251	-2.22496
-11.29638	3.94491	11.58380	0.34728
-9.77221	2.99860	10.54126	0.055096

Weight Matrix from the Hidden to the Output Layer

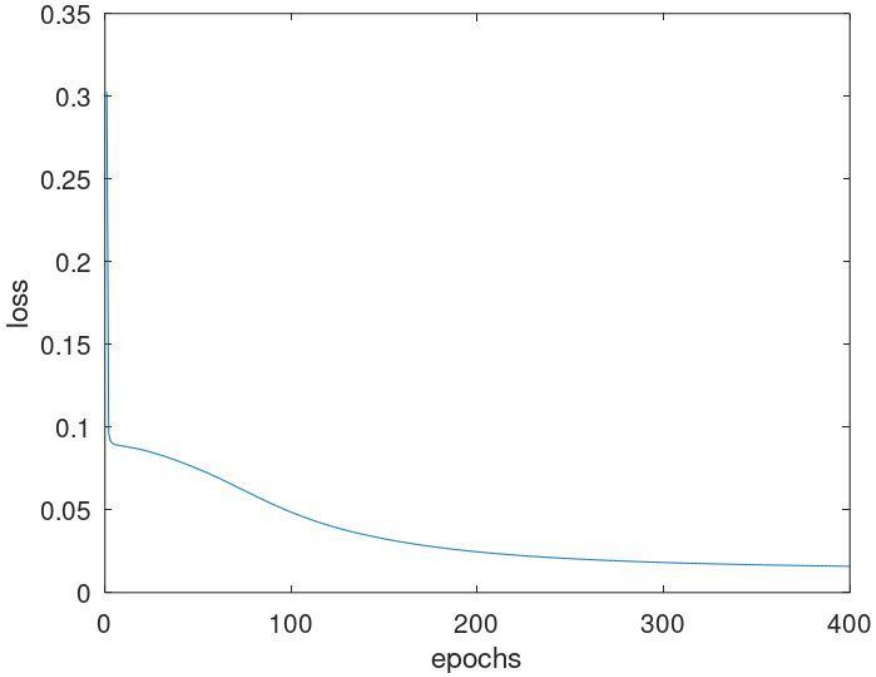
1.5896	-4.9519	-4.0163	5.0977

Fixed Point Matrix for Hidden Layer

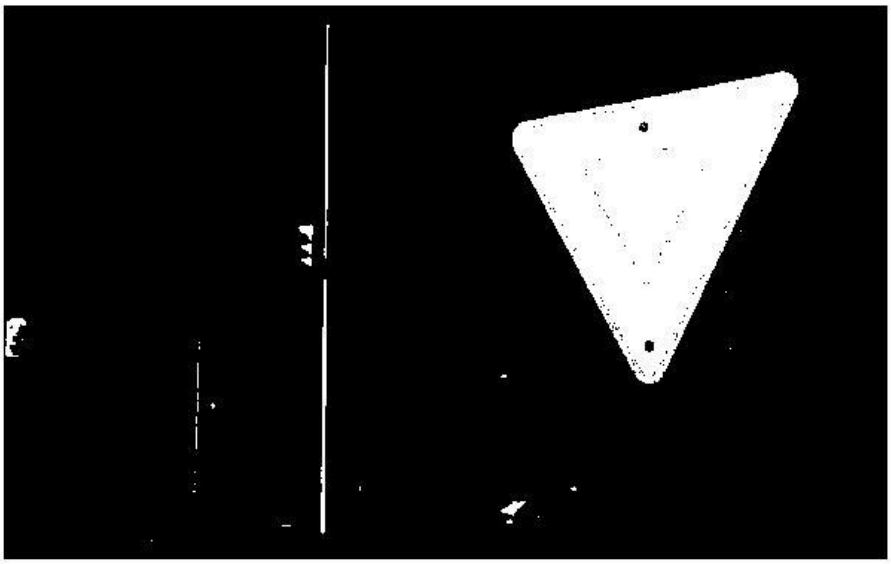
29	-45	-87	-18227
-361	126	371	2845
-313	96	337	4513

Fixed Point Matrix for Output Layer

51	-158	-129	41760



Εικόνα 16 Loss/Epochs



Εικόνα 17 Αποτελέσματα Νευρωνικού

6. Υλοποίηση του Νευρωνικού σε πλακέτα FPGA

6.1. Εισαγωγή

Η επεξεργασία εικόνας σε FPGAs καθίσταται ιδιαίτερα ελκυστική λόγω των δυνατοτήτων τους παράλληλης επεξεργασίας και της δυνατότητας επεξεργασίας σε πραγματικό χρόνο.

Οι τεχνητοί νευρώνες αποτελούν τη βάση για τα Νευρωνικά δίκτυα, που χρησιμεύουν ως οι κύριες υπολογιστικές τους μονάδες. Υπολογίζουν το σταθμισμένο άθροισμα των εισόδων τους και στη συνέχεια λαμβάνουν το αποτέλεσμα μέσω μιας συνάρτησης ενεργοποίησης.

Τα ψηφιακά συστήματα επεξεργασίας βίντεο και εικόνας απαιτούν συχνά συγχρονισμό πολλαπλών σημάτων ή εισαγωγή συγκεκριμένων χρονικών καθυστερήσεων μεταξύ τους και για τον λόγο αυτό είναι σημαντική η σωστή προετοιμασία των μεταβλητών που θα αξιοποιηθούν για να επιτευχθεί ο σωστός συγχρονισμός του συνόλου των σημάτων.

Το μοντέλο νευρωνικού δικτύου που βασίζεται σε FPGA για την επεξεργασία εικόνας RGB, όπως εξετάζεται μέσω των script VHDL, προσφέρει μια οπτική στη χρήση FPGA για την ανάπτυξη NN. Αυτή η συγχώνευση αξιοποιεί τα εγγενή οφέλη των FPGA, όπως η παράλληλη επεξεργασία, και μπορεί να φέρει επανάσταση σε εφαρμογές επεξεργασίας εικόνας και βίντεο σε πραγματικό χρόνο.

Το NN που εφαρμόστηκε στα σενάρια VHDL είναι εκείνο που εκπαιδεύτηκε στο OCTAVE.

6.2. Μεθοδολογία

Η υλοποίηση του NN σε FPGA περιλαμβάνει τα παρακάτω αρχεία:

1. neuron.vhd: Αυτό το script υλοποιεί σε γλώσσα περιγραφής υλικού έναν νευρώνα.

2. nn_rgb.vhd: Αυτό το script ορίζει την αρχιτεκτονική ανώτατου επιπέδου του νευρωνικού δικτύου με παραμέτρους εισόδου RGB και την συνολική λειτουργία επεξεργασίας των pixel.

3. control.vhd: Αυτό το κομμάτι κώδικα διασφαλίζει ότι τα σήματα ελέγχου προς το NN καθυστερούν αρκετά ώστε το FPGA να επεξεργαστεί το αμέσως προηγούμενο σήμα χωρίς να χαθούν δεδομένα.

4. sim_nn_rgb.vhd: Τέλος, το testbech, υπεύθυνο για την ανάγνωση και εγγραφή εικόνων σε μορφή αρχείου ppm και την εφαρμογή του νευρωνικού στην αρχική εικόνα.

Η οντότητα «νευρώνας» είναι το βασικό συστατικό του NN, που ενσωματώνει τη συμπεριφορά ενός μεμονωμένου νευρώνα. Διαθέτει τρία βάρη (`w1`, `w2`, `w3`), bias και τρεις εισόδους (`x1`, `x2`, `x3`). Στην ανερχόμενη ακμή κάθε κύκλου ρολογιού, υπολογίζεται το άθροισμα της εισόδου πολλαπλασιασμένο με τα αντίστοιχα βάρη τους, που προστίθενται με το bias. Αυτό το άθροισμα στη συνέχεια τροφοδοτείται σε μια συνάρτηση (μέσω αναζήτησης ROM), με αποτέλεσμα μια τιμή μεταξύ 0 και 255. Αυτή η συμπεριφορά προσομοιώνει τη συνάρτηση ενεργοποίησης, όπου αναφέρεται αναλυτικότερα στη συνέχεια, σε έναν τυπικό τεχνητό νευρώνα. Στην ουσία η ROM διατηρεί την συνάρτηση ώστε να είναι άμεσα διαθέσιμη για κλήση χωρίς να δηλώνεται εκ νέου.

Το NN (`nn_rgb`) έχει σχεδιαστεί για την επεξεργασία εικόνων RGB. Μέσα στη δομή του, η οντότητα του νευρώνα ορίζεται πολλές φορές για να δημιουργήσει τα επίπεδα εισόδου και το επίπεδο εξόδου. Τα βάρη και τα bias ορίζονται χειροκίνητα στην αρχιτεκτονική, αφού το δίκτυο είναι προεκπαιδευμένο. Αυτό το NN δρα σε δεδομένα εικονοστοιχείων RGB, τα επεξεργάζεται μέσω των νευρώνων και παράγει μια έξοδο που αντικαθιστά κάθε ένα εικονοστοιχείο με την αντίστοιχη ετικέτα, αν βρίσκεται σε σήμα οδικής κλυκλοφορίας ή όχι.

Για μια συγχρονισμένη λειτουργία επεξεργασίας εικόνας RGB, είναι απαραίτητο να υπάρχουν ακριβείς καθυστερήσεις ελέγχου. Το `control.vhd` επιτυγχάνει ακριβώς αυτή την λειτουργία εισάγοντας έναν μηχανισμό καθυστέρησης αξιοποιώντας μια μεταβλητή array για σήματα ελέγχου (`vs_in`, `hs_in`, `de_in`). Αυτή η καθυστέρηση διασφαλίζει ότι τα δεδομένα RGB υποβάλλονται σε επεξεργασία με μια συγχρονισμένη ακολουθία ελέγχου, κρίσιμης σημασίας για την ακρίβεια στην επεξεργασία.

Το αρχείο «sim_nn_rgb» χρησιμεύει ως testbench για το νευρωνικό δίκτυο, τροφοδοτώντας την ανάγνωση και εγγραφή εικόνων RGB σε μορφή ppm. Αυτό καθιστά δυνατή την προσομοίωση της απόδοσης του νευρωνικού δικτύου και την επαλήθευση της ορθότητάς του συγκρίνοντας την επεξεργασμένη εικόνα με τα αναμενόμενα αποτελέσματα.

Αναλυτικότερα η σχεδίαση VHDL με τίτλο «nn_rgb» (νευρωνικό δίκτυο RGB) ενσωματώνει την ουσία των νευρωνικών υπολογισμών στις εισερχόμενες τιμές RGB από μια ροή βίντεο.

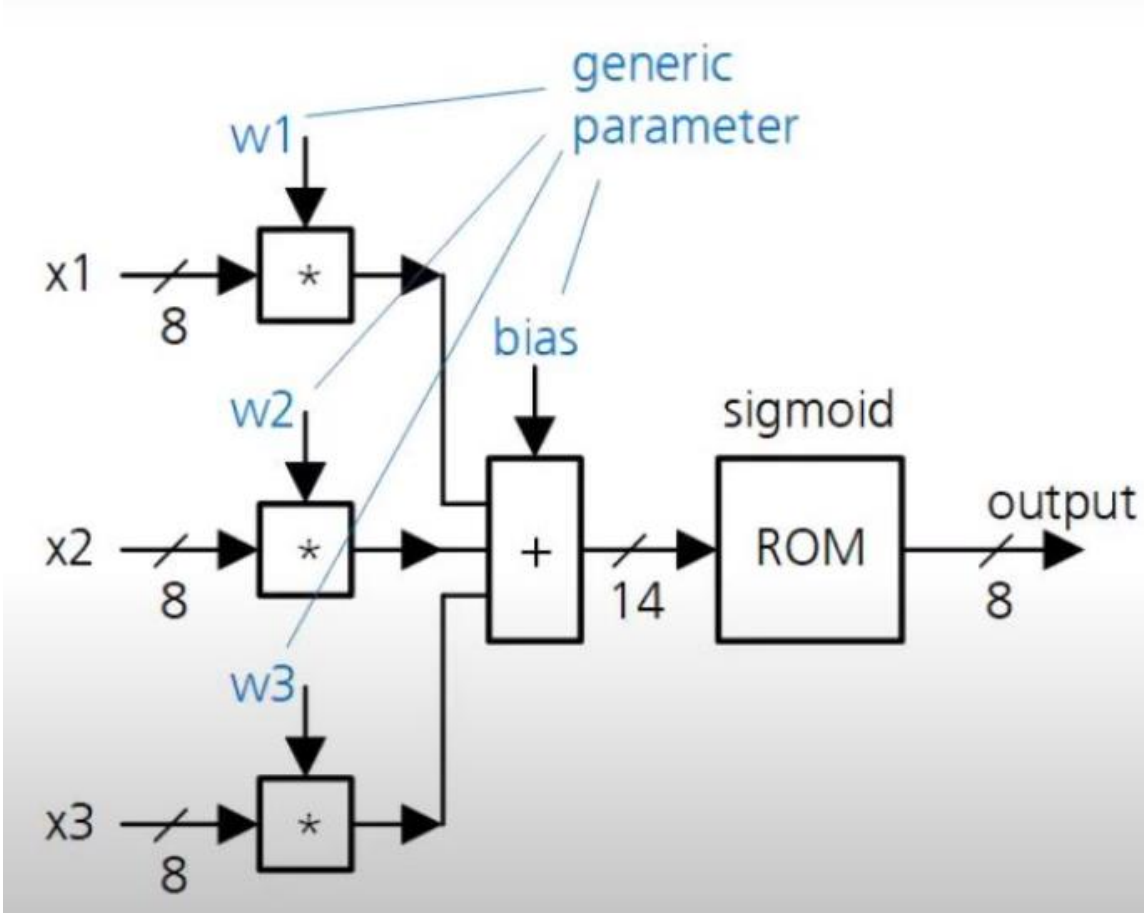
Η οντότητα «neuron» που ορίζεται στον κώδικα VHDL αποτελείται από τέσσερις γενικές μεταβλητές και τέσσερις θύρες. Οι μεταβλητές w1, w2, w3 αντιπροσωπεύουν τα βάρη που σχετίζονται με τις εισόδους του νευρώνα ενώ το bias είναι μια παράμετρος που προσαρμόζει την έξοδο ανεξάρτητα από την είσοδο. Από την άλλη πλευρά τα ports του νευρώνα αποτελούνται από το clk την είσοδο του ρολογιού και από τις 3 ακέραιες εισόδους που αντιπροσωπεύουν τις τιμές εισόδου του νευρώνα x1, x2, x3.Τέλος η έξοδος του νευρώνα είναι ένας ακέραιος αριθμός μετά την ενεργοποίηση (Εικόνα 18).

Η αρχιτεκτονική, «συμπεριφορά», περιγράφει λεπτομερώς τις υπολογιστικές διαδικασίες του νευρώνα.

Η κύρια λειτουργία ενός νευρώνα είναι να υπολογίζει το σταθμισμένο άθροισμα των εισόδων του. Αυτό λαμβάνεται από την εξίσωση $Sum \Leftarrow (w1 * x1 + w2 * x2 + w3 * x3)$. Το άθροισμα υπολογίζεται σε κάθε ανερχόμενο μέτωπο του ρολογιού.

Η διαδικασία ενεργοποίησης ενός νευρώνα στα Νευρωνικά δίκτυα τυπικά χρησιμοποιεί λειτουργίες όπως σιγμοειδής (sigmoid), hyperbolic tangent (tanh) ή rectified linear unit (ReLU). Αυτή η υλοποίηση επιλέγει μια ενεργοποίηση σιγμοειδούς, η οποία συμπιέζει τιμές μεταξύ 0 και 1. Αναλυτικότερα η σιγμοειδής ενεργοποίηση αναφέρεται στη σιγμοειδή συνάρτηση, η οποία χρησιμοποιείται συχνά ως συνάρτηση ενεργοποίησης στα Νευρωνικά δίκτυα. Ο σκοπός της συνάρτησης ενεργοποίησης είναι να προσθέσει μη γραμμικότητα στο νευρωνικό δίκτυο, επιτρέποντας του να μάθει και να προσαρμοστεί σε πιο πολύπλοκες συναρτήσεις. Η συνάρτηση αυτή είναι : $\sigma(z) = \frac{1}{1+e^{-z}}$ όπου z είναι η είσοδος της συνάρτησης ενώ e είναι η βάση του νεπέριου. Η τιμή της σιγμοειδούς συνάρτησης βρίσκεται πάντα μεταξύ 0 και 1. Για μεγάλες αρνητικές τιμές του z, η σιγμοειδής συγκλίνει προς το 0, ενώ για μεγάλες θετικές τιμές του z, συγκλίνει προς το 1. Για την αποτελεσματική υλοποίηση της σιγμοειδούς συνάρτησης, αξιοποιείται η μνήμη μόνο για ανάγνωση (ROM) που περιέχει προ-υπολογισμένες τιμές σιγμοειδούς. Το ενδιάμεσο άθροισμα αντιστοιχίζεται σε μια διεύθυνση «sumAdress»

για να ανακτηθεί η κατάλληλη σιγμοειδής τιμή.



Εικόνα 18 Νευρώνας

Η οντότητα «control» έχει δημιουργηθεί για να παρέχει μια καθορισμένη από τον χρήστη καθυστέρηση σε τρία μεμονωμένα σήματα: vs_in, hs_in και de_in.

Αρχικά ορίζεται ο ακέραιος αριθμός delay που καθορίζει την καθυστέρηση που μεταδίδεται στα σήματα. Η προεπιλεγμένη τιμή ορίστηκε στο 7.

Η αρχιτεκτονική δίνει έμφαση στη λειτουργία του μηχανισμού καθυστέρησης. Ένας νέος τύπος, delay_array, ορίζεται για να διατηρεί μια ακολουθία τιμών std_logic. Αυτός ο τύπος χρησιμοποιείται στη συνέχεια για τη δημιουργία τριών συστοιχιών σήματος vs_delay, hs_delay και de_delay για τη διατήρηση των καθυστερημένων τιμών των αντίστοιχων σημάτων εισόδου. Στο σημείο αυτό χρησιμοποιείται μια διαδικασία που επηρεάζεται στο ανερχόμενο μέτωπο του ρολογιού. Σε κάθε κύκλο ρολογιού, η τρέχουσα τιμή εισόδου αποθηκεύεται ως το πρώτο στοιχείο στον αντίστοιχο πίνακα καθυστέρησης. Στη συνέχεια, οι υπάρχουσες τιμές στον πίνακα μετατοπίζονται προς τα δεξιά. Αυτή η ενέργεια προσομοιώνει τη συμπεριφορά ενός καταχωρητή μετατόπισης. Ο βρόχος for διασφαλίζει ότι οι τιμές διαδίδονται μέσω του πίνακα, δημιουργώντας την

καθυστέρηση. Η διάρκεια αυτής της καθυστέρησης καθορίζεται από το μέγεθος της διάταξης καθυστέρησης, η οποία ορίζεται από τη γενική καθυστέρηση.

Τα καθυστερημένα σήματα εξόδου είναι απλώς το τελευταίο στοιχείο των αντίστοιχων συστοιχιών καθυστέρησης, οι οποίες θα έχουν διατηρήσει την τιμή εισόδου για τη διάρκεια που καθορίζεται από τη generic μεταβλητή για την καθυστέρηση.

Ο σχεδιασμός «control» εισάγει αποτελεσματικά μια σταθερή καθυστέρηση στα ψηφιακά σήματα χρησιμοποιώντας έναν απλό και διαισθητικό μηχανισμό που βασίζεται σε μια ιδέα καταχωρητή μετατόπισης. Ο γενικός σχεδιασμός του εξασφαλίζει ευελιξία, επιτρέποντας στον χρήστη να ορίσει την επιθυμητή καθυστέρηση.

Η μονάδα ελέγχου χρησιμεύει ως μια αποτελεσματική ψηφιακή γραμμή καθυστέρησης με μια παράμετρο καθυστέρησης που ορίζεται από το χρήστη. Η γενική και αρθρωτή δομή του επιτρέπει την προσαρμογή και την επεκτασιμότητα. Οι υλοποιήσεις τέτοιων σχεδίων είναι ζωτικής σημασίας σε συστήματα ψηφιακών βίντεο, διασφαλίζοντας συγχρονισμένη επεξεργασία και μετάδοση δεδομένων.

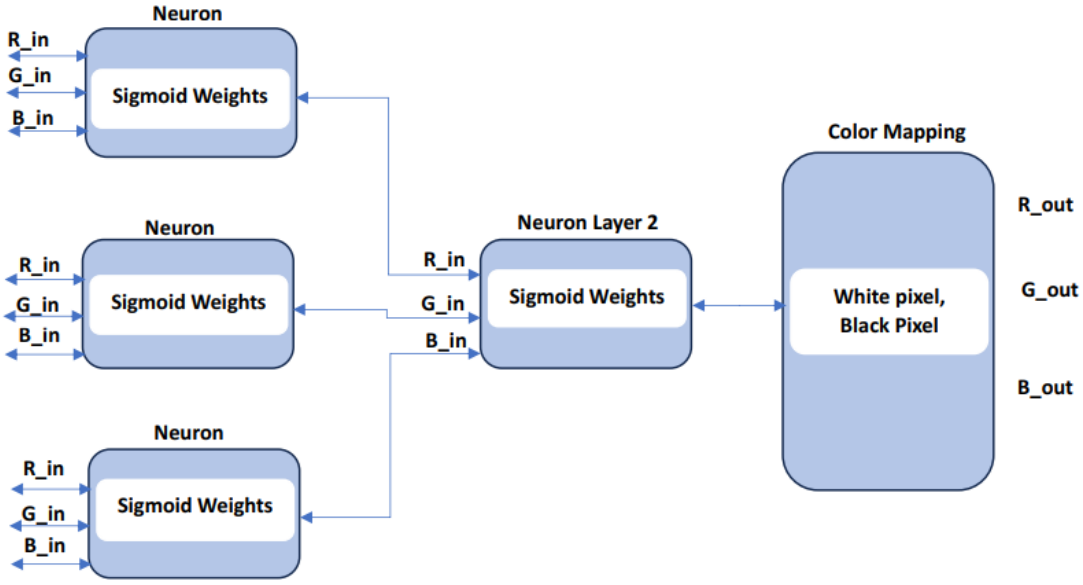
Με σκοπό την καλύτερη κατανόηση της ενότητας αυτής, αρχικά πρέπει να γίνει αναφορά στα σήματα που χρησιμοποιούνται και ανάγονται σε τρεις κατηγορίες, σήματα ελέγχου, σήματα εισόδου και σήματα εξόδου. Τα σήματα ελέγχου περιλαμβάνουν το ρολόι (clk), ένα σήμα ανεστραμμένης επαναφοράς (reset_n) και ένα σήμα ενεργοποίησης (enable_in), τα σήματα εισόδου περιλαμβάνουν σήματα συγχρονισμού (vs_in, hs_in και de_in) και στοιχεία RGB του εικονοστοιχείου (r_in, g_in και b_in) και τέλος τα σήματα εξόδου με τα επεξεργασμένα εικονοστοιχεία.

Τα κύρια στοιχεία της αρχιτεκτονικής είναι τα σήματα (επαναφορά, ενεργοποίηση, vs_0, hs_0, de_0, r_0, g_0, b_0) είναι είσοδοι προσωρινής αποθήκευσης τα οποία διασφαλίζοντας την ακεραιότητα των δεδομένων σε όλο τον υπολογιστικό κύκλο. Βασικό και ίσως το σημαντικότερο κομμάτι είναι οι νευρώνες. Η οντότητα του νευρώνα παρουσιάζεται τέσσερις φορές. Οι πρώτες τρεις περιπτώσεις (hidden0, hidden1 και hidden2) αντιπροσωπεύουν το κρυφό στρώμα του νευρωνικού δικτύου, ενώ η έξοδος0 αντιπροσωπεύει το επίπεδο εξόδου. Κάθε νευρώνας εκτελεί ένα σταθμισμένο άθροισμα των εισόδων του, ακολουθούμενο από μια συνάρτηση ενεργοποίησης. Η έξοδος του δικτύου ορίζεται στην τιμή του 127, παράγοντας ένα αποτέλεσμα όλων των εικονοστοιχείων '1' ή όλων των '0' για r_out, g_out και b_out αντίστοιχα.

Η ενότητα nn_rgb υιοθετεί μια απλή αλλά αποτελεσματική αρχιτεκτονική νευρωνικών δικτύων για επεξεργασία RGB με χρήση Perceptron πολλαπλών επιπέδων. Η δομή σχεδίασης, που αποτελείται από ένα στρώμα εισόδου και ένα στρώμα εξόδου,

είναι χαρακτηριστική ενός MLP (Εικόνα 19). Το στρώμα εισόδου περιλαμβάνει τρεις νευρώνες και το στρώμα εξόδου αποτελείται από έναν μόνο νευρώνα. Το νευρωνικό δίκτυο έχει εκπαιδευτεί εξωτερικά, δεδομένων των βαρών και των προκαταλήψεων που εισάγονται χειροκίνητα στους γενικούς χάρτες των νευρώνων. Η σχεδίαση αυτή διασφαλίζει την επεξεργασία των εικονοστοιχείων σε πραγματικό χρόνο.

Η ενότητα nn_rgb VHDL παρουσιάζει έναν αποτελεσματικό συνδυασμό αρχών νευρωνικών δικτύων και ψηφιακού σχεδιασμού. Με την επεξεργασία των τιμών RGB και την παροχή επεξεργασμένων εξόδων, υπογραμμίζει τη δύναμη των FPGA στις εφαρμογές επεξεργασίας εικόνας.



Εικόνα 19 Νευρωνικό Δίκτυο

Τελευταίο μέρος της υλοποίησης είναι η προσομοίωση του νευρικού δικτύου (`nn_rgb`) που επεξεργάζεται σε δεδομένα εικόνας RGB. Το νευρωνικό δίκτυο επεξεργάζεται ένα μόνο εικονοστοιχείο RGB κάθε φορά και αυτή η προσομοίωση έχει ρυθμιστεί για την επεξεργασία μιας εικόνας σε μορφή PPM (Portable Pixmap).

Η διαδικασία της προσομοίωσης ξεκινάει διαβάζοντας μια εικόνα PPM (`first_image.ppm`) pixel προς pixel και την τροφοδοτεί στην οντότητα `nn_rgb`. Για κάθε pixel, οι τιμές RGB διαβάζονται από το αρχείο και μεταβιβάζονται στο DUT (Device Under Test). Η προσομοίωση χωρίζεται σε δύο κύριες διαδικασίες:

`stimuli_process`: διαβάζει το αρχείο εικόνας εισόδου και το τροφοδοτεί pixel-pixel

στο DUT.

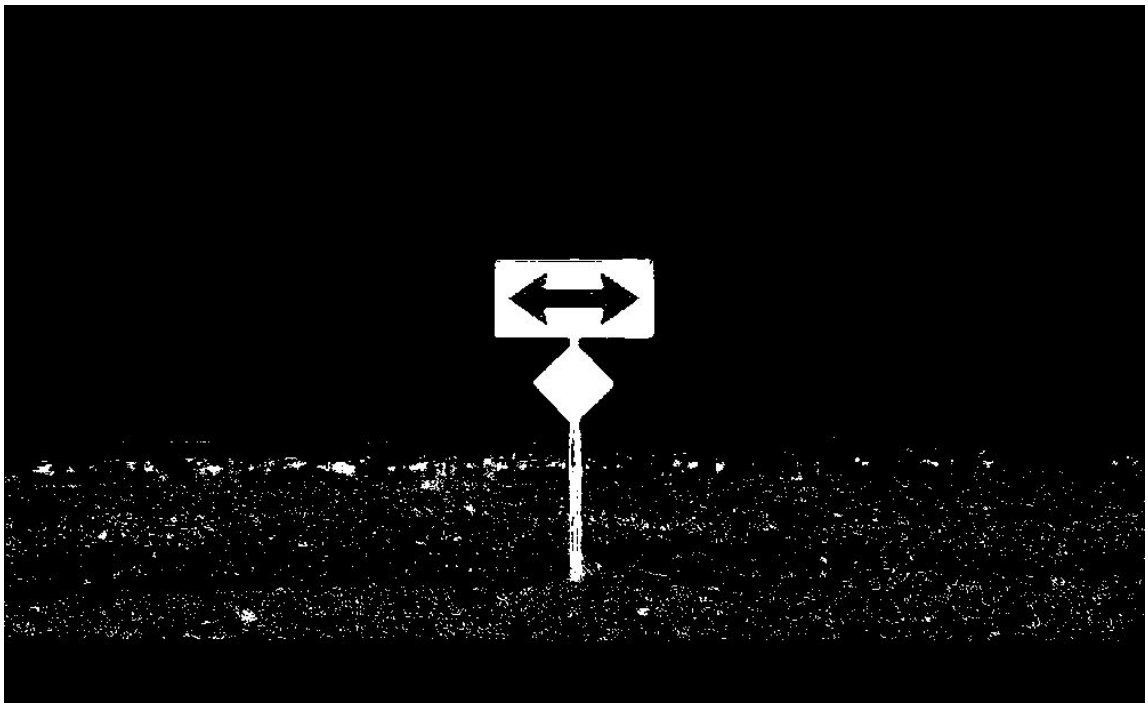
`response_process`: καταγράφει την έξοδο του DUT και το εγγράφει σε άλλο αρχείο PPM (`image_response.ppm`).

Η μορφή αρχείου εικόνας που χρησιμοποιείται εδώ είναι PPM. Είναι μια απλή μορφή εικόνας όπου τα δεδομένα pixel αποθηκεύονται ως ακατέργαστες τιμές RGB. Η προσομοίωση διαβάζει αυτό το αρχείο, το επεξεργάζεται μέσω του νευρωνικού δικτύου και εγγράφει την επεξεργασμένη εικόνα σε μορφή PPM. Η προσομοίωση θα εκτελεστεί έως ότου όλα τα εικονοστοιχεία από την εικόνα εισόδου έχουν υποστεί επεξεργασία και εγγραφή στην εικόνα εξόδου.

Όσον αφορά τη λειτουργικότητα, αυτή η ρύθμιση προσομοιώνει την επίδραση του νευρωνικού δικτύου σε μια εικόνα RGB. Το δίκτυο `nn_rgb` ουσιαστικά λειτουργεί ως φίλτρο για κάθε pixel στην εικόνα. Το πώς μετατρέπει κάθε pixel εξαρτάται από τα βάρη και τις προκαταλήψεις (bias) των νευρώνων. Η επεξεργασμένη εικόνα θα αποθηκευτεί στο `image_response.ppm` και μπορεί να οπτικοποιηθεί χρησιμοποιώντας οποιοδήποτε λογισμικό/εργαλείο που υποστηρίζει τη μορφή PPM.

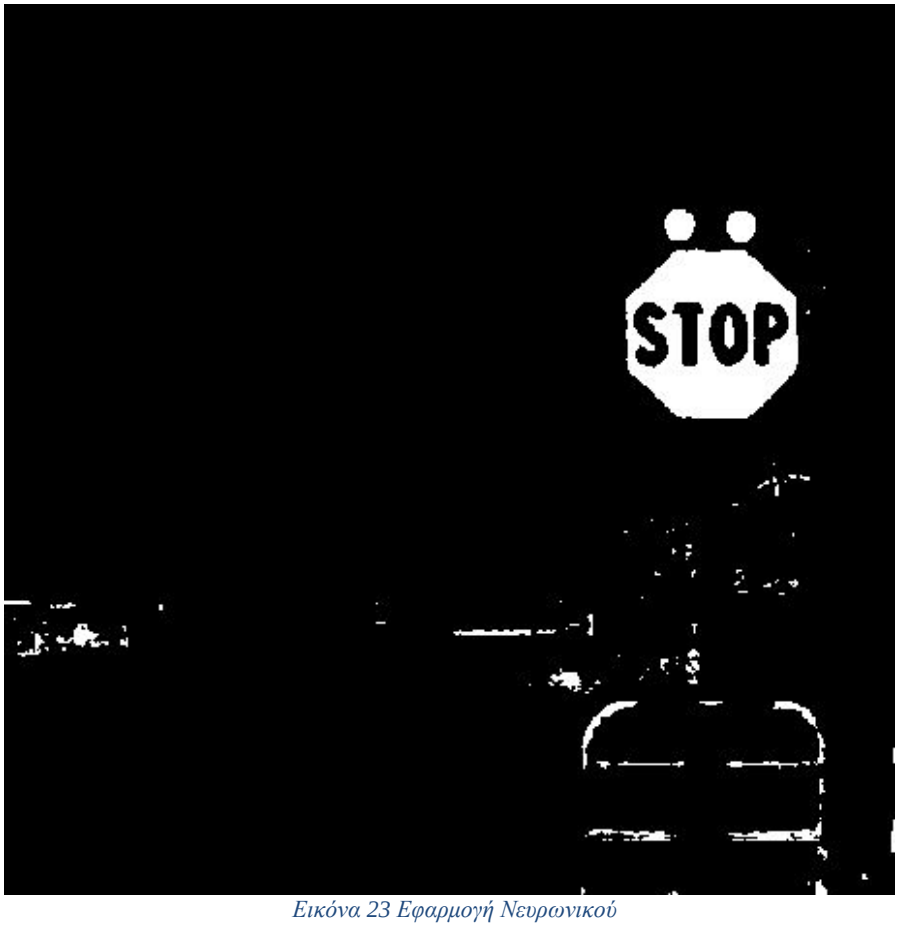
Εφόσον έχει αναλυθεί η συνολική διαδικασία της εκπαίδευσης και της εφαρμογής του NN, εφαρμόστηκε σε δύο εικόνες (Εικόνα 20, 22) ώστε να φανεί η αποτελεσματικότητα του. Παρατηρείται ότι δεν είναι απόλυτος ο εντοπισμός της εικόνας αλλά το NN έχει σημαδέψει και άλλα pixel της εικόνας το οποίο οφείλεται στον τρόπο εκπαίδευσης του NN και τα αποτελέσματα φαίνονται στις εικόνες 21 και 23.

Εικόνα 20 Εικόνα πριν την επεξεργασία



Εικόνα 21 Εφαρμογή Νευρωνικού

Εικόνα 22 Εικόνα πριν την επεξεργασία



Εικόνα 23 Εφαρμογή Νευρωνικού

7. Συμπεράσματα

Κατά τη διάρκεια αυτής της διατριβής, διερευνήθηκε η εφαρμογή νευρωνικών δικτύων σε Field-Programmable Gate Arrays (FPGAs). Το εξελισσόμενο τοπίο της τεχνητής νοημοσύνης απαιτεί λύσεις υλικού που μπορούν να προσαρμοστούν, να κλιμακωθούν και να βελτιστοποιηθούν για να ανταποκριθούν στις περίπλοκες απαιτήσεις των αλγορίθμων νευρωνικών δικτύων. Τα FPGA, με τον μοναδικό συνδυασμό αναδιαμόρφωσης, δυνατοτήτων παράλληλης επεξεργασίας και ενεργειακής απόδοσης, έχουν αναδειχθεί ως μια ελκυστική πλατφόρμα για αυτό το εγχείρημα.

Μέσα από διάφορες υλοποιήσεις και μελέτες περίπτωσης, έχουμε αποδείξει τα βαθιά πλεονεκτήματα που μπορούν να προσφέρουν οι αρχιτεκτονικές που βασίζονται σε FPGA για υπολογισμούς νευρωνικών δικτύων. Όχι μόνο επιτρέπουν προσαρμοσμένες βελτιστοποιήσεις, αλλά παρέχουν επίσης ευκαιρίες για αναδιαμορφώσεις on the fly, καλύπτοντας τις δυναμικές ανάγκες διαφορετικών μοντέλων νευρωνικών δικτύων.

Ωστόσο, το ταξίδι της ενοποίησης των FPGA και των νευρωνικών δικτύων δεν είναι χωρίς προκλήσεις. Οι αλλαγές στην πολυπλοκότητα του σχεδιασμού, τα παραδείγματα προγραμματισμού και την επεκτασιμότητα απαιτούν προσεκτική σχεδίαση. Καθώς ο τομέας εξελίσσεται, η ανάπτυξη ισχυρών εργαλείων, πλαισίων εργασίας και βέλτιστων πρακτικών θα είναι ζωτικής σημασίας για να διασφαλιστεί ότι τα FPGA είναι προσβάσιμα και αποτελεσματικά για ευρύτερες εφαρμογές τεχνητής νοημοσύνης.

Κλείνοντας, ο συνδυασμός νευρωνικών δικτύων και FPGA δίνει μια πολλά υποσχόμενη εικόνα για το μέλλον του υλικού AI. Καθώς βρισκόμαστε στο κατώφλι μιας νέας εποχής τεχνολογικής εξέλιξης, τα FPGA έχουν καθιερωθεί ως μια βασική επιλογή στο υπολογιστικό τοπίο, έτοιμη να επηρεάσει σημαντικά την τροχιά της έρευνας και της εφαρμογής των νευρωνικών δικτύων.

8. Μελλοντικές Βελτιώσεις

Η μελέτη ξεκίνησε την εξερεύνηση της χρήσης νευρωνικών δικτύων σε πλατφόρμες FPGA για την ανίχνευση οδικών σημάτων από δεδομένα εικόνας. Υπό το φως των πειραματικών αποτελεσμάτων και της βιβλιογραφίας που έγινε ανασκόπηση, προσφέρονται οι ακόλουθες συστάσεις για τη βελτίωση του βάθους και του εύρους αυτής της έρευνας.

Ένα ολοκληρωμένο σύνολο μετρήσεων πέρα από την accuracy, όπως precision, recall, and F1-score, θα παρείχε μια ολιστική εικόνα για την απόδοση του μοντέλου. Μια σχολαστική ανατομή (analysis) των ψευδών θετικών και αρνητικών είναι επιτακτική.

Η επιβεβαίωση της ικανότητας του συστήματος στην ανίχνευση σημαδιών από ροές βίντεο ή ροές βίντεο σε πραγματικό χρόνο θα προωθούσε τις πρακτικές του συνέπειες.

Εφαρμογή της μονάδας ανίχνευσης οδικών πινακίδων με υφιστάμενα συστήματα πλοήγησης ή μελλοντικές τεχνολογίες αυτόνομων οχημάτων.

Τέλος θα μπορούσε να σχεδιαστεί μια διαισθητική διεπαφή που να υποστηρίζεται από μηχανισμό ανάδρασης σε πραγματικό χρόνο και δυνατότητες ενημέρωσης συστήματος και των χρηστών της διεπαφής αυτής.

Η ενσωμάτωση αυτών των συστάσεων θα μπορούσε ενδεχομένως να βελτιώσει την ακαδημαϊκή συμβολή αυτής της μελέτης και να ενισχύσει τη μεταφραστική της δυνατότητα σε απτές εφαρμογές.

9. Ανάλυση επίλυσης / Ανάλυση κώδικα

Γενική Περιγραφή **control**:
Αυτός ο κώδικας αναπαριστά ένα μηχανισμό καθυστέρησης για τρία σήματα (vs_in, hs_in, de_in). Η καθυστέρηση καθορίζεται από μια μεταβλητή με το όνομα "delay", η οποία ορίζεται κατά την αρχικοποίηση του εξαρτήματος.

Ανάλυση γραμμή προς γραμμή:
1. "library IEEE;" - βιβλιοθήκη IEEE

2. "use IEEE.STD_LOGIC_1164.ALL;" - Αυτό εισάγει όλους τους απαραίτητους ορισμούς για τους τύπους std_logic και std_logic_vector.

3. "use IEEE.NUMERIC_STD.ALL;" - Χρησιμοποιείται για να ενσωματώσει αριθμητικές λειτουργίες που θα χρησιμοποιηθούν παρακάτω

5-15. "entity control is ... end control;" - Ορίζει τη διεπαφή του εξαρτήματος "control". Διαθέτει μια ακέραιη μεταβλητή "delay", η οποία ορίζεται σε 7. Το εξάρτημα έχει πέντε σήματα εισόδου (clk, reset, vs_in, hs_in, de_in) και τρία σήματα εξόδου (vs_out, hs_out, de_out).

17. "architecture behave of control is" - Αρχή της αρχιτεκτονικής με το όνομα "behave" για το εξάρτημα "control".

20-23. "type delay_array is ... signal de_delay : delay_array;" - Ορίζει έναν προσαρμοσμένο τύπο δεδομένων "delay_array", που είναι ένας πίνακας std_logic με μήκος που καθορίζεται από το γενικό "delay". Τρία σήματα (vs_delay, hs_delay, de_delay) δηλώνονται από αυτόν τον τύπο για την αποθήκευση ‘delayed version’ των σημάτων εισόδου.

27. "process ... end process;" - Αυτό είναι ένα μπλοκ διεργασίας που αντιδρά στην άνοδο του σήματος "clk" (rising edge).

32-34. "vs_delay(1) <= vs_in; ... de_delay(1) <= de_in;" - Αυτές οι γραμμές ενημερώνουν τον πρώτο στοιχείο κάθε πίνακα με την τρέχουσα τιμή του αντίστοιχου σήματος εισόδου.

37-41. "for i in 2 to delay loop ... end loop;" - Αυτός ο βρόχος μετακινεί τις τιμές στους πίνακες καθυστέρησης, εισάγοντας αποτελεσματικά ένα delay. Η πιο πρόσφατη τιμή βρίσκεται στο δείκτη 1, και καθώς προχωράμε στον πίνακα μετά από τον μέγιστο δείκτη (που καθορίζεται από την "delay"), βλέπουμε παλαιότερες τιμές.

45-50. "vs_out <= vs_delay(delay); ... de_out <= de_delay(delay);" - Αναθέτει στα σήματα εξόδου (vs_out, hs_out, de_out) τις παλαιότερες τιμές από τους πίνακες καθυστέρησης, που αποτελούν αποτελέσματα από την είσοδο "delay" ρολογιών κύκλων πριν. Εάν το γενικό "delay" δεν καθοριστεί κατά την αρχικοποίηση, προεπιλέγεται στο 7, προσδιορίζοντας έτσι μια καθυστέρηση 7 κύκλων ρολογιού.
Το συγκεκριμένο module ουσιαστικά καθυστερεί τα σήματα εισόδου (`vs_in`, `hs_in`, and `de_in`) κατά έναν συγκεκριμένο αριθμό κύκλων ρολογιού που ορίζεται από την μεταβλητή delay και δίνει όρισμα στα σήματα εξόδου.

Γενική Περιγραφή **neuron**:
Αναπαριστά ένα νευρώνα από ένα νευρικό δίκτυο. Υπολογίζει τον γινόμενο των εισόδων του. Πολλαπλασιάζει κάθε από τις τρεις ακέραιες εισόδους του (`x1`, `x2`, `x3`) με το αντίστοιχο βάρος (`w1`, `w2`, `w3`), τις προσθέτει όλες, προσθέτει μια τιμή `bias`, και στη συνέχεια περνά το αποτέλεσμα από μια συνάρτηση ενεργοποίησης sigmoid για να παράγει την έξοδό του.

Ανάλυση γραμμή προς γραμμή:
1-3. Δηλώσεις βιβλιοθήκης για την βιβλιοθήκη IEEE και τους τύπους δεδομένων.
5-16. `entity neuron is ... end neuron;`
- Ορίζονται τέσσερις γενικές μεταβλητές: `w1`, `w2`, `w3` και `bias`, τα οποία χρησιμοποιούνται για να παραμετροποιήσουν τα βάρη και το bias του νευρώνα.
- Οι είσοδοι περιλαμβάνουν ένα σήμα ρολογιού `clk` και τρεις ακέραιες εισόδους που αντιπροσωπεύουν τις τιμές εισόδου του νευρώνα.
- Ορίζεται μια ακέραια έξοδος `output`, που θα είναι η υπολογισμένη τιμή του νευρώνα μετά την εφαρμογή των βαρών, του bias και της συνάρτησης ενεργοποίησης.

18. `architecture behave of neuron is`
- Η αρχή της αρχιτεκτονικής με το όνομα `behave` για το εξάρτημα `neuron`.

20-22. Δηλώνονται τρία εσωτερικά σήματα:
- `sum`: Για να αποθηκεύει τον γινόμενο των βαρών εισόδου συν το bias.
- `sumAdress`: Μια 16-bit `std_logic_vector` αναπαράσταση της τιμής `sum`, που θα χρησιμοποιηθεί ως διεύθυνση για την ROM (Read Only Memory) που υλοποιεί τη συνάρτηση sigmoid.
- `afterActivation`: Ένα 8-bit τιμή που αντιπροσωπεύει την έξοδο της συνάρτησης sigmoid.

26-39. Το μπλοκ διεργασίας αντιδρά στην άνοδο του σήματος `clk`.
- Μέσα στη διαδικασία,

υπολογίζει τον γινόμενο των βαρών εισόδου συν το bias και τον αναθέτει στο `sum`.
- Στη συνέχεια, ελέγχει την τιμή του `sum` και την περιορίζει στο εύρος `-32768 έως 32767`. Η τιμή στη συνέχεια μετατοπίζεται και μετατρέπεται σε μια 16-bit `std_logic_vector`, η οποία αποθηκεύεται στο `sumAdress`.

42-45. Δημιουργείται ένα παράδειγμα του εξαρτήματος `sigmoid_IP`. Αυτό το εξάρτημα χρησιμοποιεί μια ROM για να προσεγγίσει τη συνάρτηση sigmoid. Τα σχετικά δυαδικά ψηφία διεύθυνσης του `sumAdress` περνιούνται στη θύρα `address`, και το αποτέλεσμα αποθηκεύεται στο `afterActivation`.

48. Μετατρέπει την 8-bit τιμή `afterActivation` από `std_logic_vector` σε ακέραιο, ο οποίος ανατίθεται στη θύρα `output`.

Αυτό το module, όταν του δίνονται τρεις ακέραιες εισόδους και τα αντίστοιχα βάρη τους συν το bias, υπολογίζει την τιμή του νευρώνα πολλαπλασιάζοντας και αθροίζοντας τις εισόδους, και στη συνέχεια περνά το αποτέλεσμα από μια συνάρτηση ενεργοποίησης sigmoid, και τελικά εξάγει μια αναπαράσταση ακεραίου της τιμής του νευρώνα.

Γενική Περιγραφή **sigmoid_IP**:

Το `sigmoid_IP`, δεσμεύει μια ROM που περιέχει μια προεπιλεγμένη συνάρτηση sigmoid. Αυτή η ROM υλοποιείται χρησιμοποιώντας το `altsyncram` megafunction της Intel και οι τιμές στη ROM προέρχονται από ένα αρχείο αρχικοποίησης μνήμης (MIF) με το όνομα `sigmoid_14_bit.mif`.

1. Σχόλια:
2. Δηλώσεις Βιβλιοθήκης:
   - Οι `LIBRARY ieee;` και `LIBRARY altera_mf;` δηλώνουν τις απαραίτητες βιβλιοθήκες.
   - Οι `USE ieee.std_logic_1164.all;` και `USE altera_mf.altera_mf_components.all;` επιτρέπουν τη χρήση των standard logic types και των εξαρτημάτων που είναι ειδικά για τα προϊόντα της Altera.

3. Δήλωση Entity:
   - `ENTITY sigmoid_IP IS ... END sigmoid_IP;`: Δηλώνει το module.
     - `address`: 14-bit είσοδος αντιπροσωπεύει ένα δείκτη στο ROM.
     - `clock`: Είσοδος ρολογιού με προεπιλεγμένη τιμή '1'.
     - `q`: 8-bit έξοδος, αντιπροσωπεύει την τιμή που ανακτάται από το ROM.

4. Αρχιτεκτονική:
   - Η αρχιτεκτονική ονομάζεται `SYN` και ανήκει στο εξάρτημα `sigmoid_ip`.
   - Δηλώνεται ένα σήμα με το όνομα `sub_wire0`, που είναι ένα 8-bit σήμα, χρησιμοποιείται ως ενδιάμεσο μεταξύ της ROM και της εξόδου.
   - Η έξοδος της ROM, `q_a`, συνδέεται με το `sub_wire0`, το οποίο συνδέεται άμεσα με την έξοδο `q` του εξαρτήματος.

5. Υλοποίηση του ROM:
   - Η επικεφαλίδα `altsyncram_component: altsyncram` αναπαριστά τη ROM. Διαθέτει διάφορες παραμέτρους (γνωστές ως generics στο VHDL) για να ρυθμίσει τη συμπεριφορά του:
     - `init_file => "sigmoid_14_bit.mif"`: Καθορίζει το αρχείο MIF που αρχικοποιεί το περιεχόμενο της ROM.
     - `operation_mode => "ROM"`: Ρυθμίζει το εξάρτημα να λειτουργεί ως ROM.
     - `numwords_a => 16384`: Υποδηλώνει ότι η ROM μπορεί να αποθηκεύσει 16.384 λέξεις, που αντιστοιχούν σε μια 14-bit διεύθυνση.
     - `width_a => 8`: Κάθε λέξη στη ROM είναι 8 bits.

6. Αντιστοίχιση Θυρών:
   - Αυτή η ενότητα συνδέει τις εισόδους και εξόδους του εξαρτήματος ROM με τις θύρες και τα εσωτερικά σήματα του εξαρτήματος.
     - Το `address_a` (είσοδος διεύθυνσης της ROM) συνδέεται με τη θύρα `address` του εξαρτήματος.
     - Το `clock0` (είσοδος ρολογιού της ROM) συνδέεται με τη θύρα `clock` του εξαρτήματος.
     - Το `q_a` (έξοδος δεδομένων της ROM) συνδέεται με το σήμα `sub_wire0`.

7. Πληροφορίες Ανάκτησης:
   - Αυτή η ενότητα περιέχει δεδομένα σχετικά με το παραγόμενο εξάρτημα και τον

τρόπο που ρυθμίστηκε στο Megafunction Wizard.

Το `sigmoid_IP` παρέχει μια υλοποίηση της συνάρτησης sigmoid βασισμένη σε ROM, όπου οι τιμές της συνάρτησης έχουν προεπιλεγεί και αποθηκεύονται στη ROM. Δεδομένης μιας `address` ως εισόδου, που αντιπροσωπεύει μια τιμή για την οποία θα υπολογιστεί η συνάρτηση sigmoid, το εξάρτημα εξάγει την αντίστοιχη τιμή sigmoid από το ROM στη θύρα `q`.

Γενική Περιγραφή **nn_rgb**:
Ο κώδικας ένα νευρικό δίκτυο που επεξεργάζεται δεδομένα εικόνας RGB.

Entity: `nn_rgb`

Η κύρια διεπαφή αυτού του εξαρτήματος δηλώνεται μέσω του `entity nn_rgb`. Αυτό το entity έχει αρκετές θύρες εισόδου και εξόδου:

Είσοδοι:
- `clk`: Είσοδος ρολογιού.
- `reset_n`: Σήμα επαναφοράς.
- `enable_in`: Τρία διακόπτες ενεργοποίησης.
- Σήματα εισόδου βίντεο (`vs_in`, `hs_in`, `de_in`, `r_in`, `g_in`, `b_in`) που αντιπροσωπεύουν το συγχρονισμό (sync) και τα κανάλια χρώματος κόκκινο, πράσινο και μπλε μιας εικόνας.

Εξόδοι:
- Σήματα εξόδου βίντεο (`vs_out`, `hs_out`, `de_out`, `r_out`, `g_out`, `b_out`), που αντιστοιχούν στα είσοδος σήματα συγχρονισμού και RGB.
- `clk_o`: Έξοδος ρολογιού.

Αρχιτεκτονική: `behave`
Εδώ καθορίζεται η συμπεριφορά του εξαρτήματος.

Βασικά Στοιχεία:
  - Καταγράφει τα εισερχόμενα σήματα (`reset`, `enable`, `vs_0`, `hs_0`, `de_0`, `r_0`, `g_0`, `b_0`).
  - Τα εισερχόμενα σήματα RGB μετατρέπονται από 8-bit τιμές σε ακέραιους για επεξεργασία.

2. Νευρώνες:
  - Αυτοί είναι οι θεμελιώδεις οικοδομικοί του νευρικού δικτύου. Επεξεργάζονται τα δεδομένα RGB.
  - Τρεις νευρώνες (`hidden0`, `hidden1`, `hidden2`) και ένας νευρώνας (`output0`).
  - Κάθε νευρώνας λαμβάνει τα δεδομένα RGB (ή δεδομένα από προηγούμενους νευρώνες) και ορισμένα προκαθορισμένα βάρη και παραμέτρους για τη δημιουργία μιας εξόδου.

3. Μονάδα Ελέγχου:
  - Εξασφαλίζει τον συγχρονισμό για την επεξεργασία της εικόνας.
  - Καθυστερεί τα σήματα για ένα συγκεκριμένο χρονικό διάστημα (που καθορίζεται από το γενικό `delay`).

Διαδικασίες:

1. Καταγράφει τα εισερχόμενα σήματα στην ανόδου της ακμής του ρολογιού.

2.
   - Με βάση την έξοδο του νευρικού δικτύου, τα κανάλια RGB είτε ενεργοποιούνται είτε απενεργοποιούνται.
   - Εάν η έξοδος είναι μεγαλύτερη από 127, τα κανάλια RGB ορίζονται 1, διαφορετικά 0.

Τέλος, το σήμα `clk_o` ακολουθεί απλώς το εισερχόμενο `clk`
Αυτό το VHDL εξάρτημα αντιπροσωπεύει ένα απλό feed-forward νευρικό δίκτυο με τρεις κρυφούς νευρώνες που επεξεργάζεται δεδομένα. Ανάλογα με την έξοδο του νευρικού δικτύου, τα κανάλια RGB είναι είτε ενεργοποιημένα (λευκό) είτε απενεργοποιημένα (μαύρο).

Γενική Περιγραφή **sim_nn_rgb**:
Αυτός ο κώδικας είναι ένα testbench με το όνομα `sim_nn_rgb`. Ο κύριος σκοπός του testbench είναι να προσομοιώσει τη συμπεριφορά του `nn_rgb` σε δεδομένα εικόνας και να παρακολουθήσει την έξοδο.

Βασικά Χαρακτηριστικά και Λειτουργίες:

1. Είσοδος και Έξοδος από Αρχείο:
   - Διαβάζει μια εικόνα στη μορφή PPM από ένα αρχείο με το όνομα "first_image.ppm".
   - Γράφει τα επεξεργασμένα αποτελέσματα σε ένα άλλο αρχείο με το όνομα "image_response.ppm".

2. Ρύθμιση της Προσομοίωσης:
   - Δημιουργεί ένα σήμα ρολογιού με συχνότητα 100MHz.
   - Επαναφέρει και αρχικοποιεί την μονάδα σχεδιασμού που δοκιμάζεται (DUT).

3. Τροφοδότηση του Σχεδιασμού:
   - Διαβάζει τιμές RGB pixel από το αρχείο εισόδου και τις τροφοδοτεί στη μονάδα σχεδιασμού `nn_rgb`, προσομοιώνοντας πώς θα επεξεργαζόταν πραγματικά δεδομένα εικόνας σε πραγματικό χρόνο.
   - Τα pixel διαβάζονται γραμμή προς γραμμή και τροφοδοτούνται στο DUT με τα αντίστοιχα σήματα ελέγχου (`de_in`, `hs_in`, `vs_in`).

4. Καταγραφή της Εξόδου:
   - Παρακολουθεί τα σήματα εξόδου του `nn_rgb`.
   - Όταν διατίθενται έγκυρα δεδομένα (όπως υποδεικνύεται από το σήμα `de_out`), καταγράφονται οι επεξεργασμένες RGB τιμές και γράφονται στο αρχείο εξόδου PPM.

Κύρια Στοιχεία:

- Σταθερές και Σήματα: Καθορίζουν τις λειτουργικές παραμέτρους της προσομοίωσης, όπως τα ονόματα αρχείων εισόδου και εξόδου.

- Ενσωμάτωση της Μονάδας Σχεδιασμού (DUT): Το `nn_rgb` module ενσωματώνεται μέσα στο testbench για να επιτρέψει την αλληλεπίδραση.

- Διαδικασία Stimuli: Υπεύθυνη για την ανάγνωση του αρχείου εισόδου, την ρύθμιση του περιβάλλοντος προσομοίωσης και την τροφοδότηση δεδομένων στο DUT.

- Διαδικασία Response: Καταγράφει τα δεδομένα εξόδου από το DUT και τα γράφει στο αρχείο εξόδου.

Αυτή η ενότητα παρέχει σχόλια σχετικά με το σεναρίου και τον σκοπό της συνάρτησης «trainNetwork».

Υπογραφή συνάρτησης:

function[trainedNetwork, cost_log, trainingSetAccuracy, validationSetAccuracy] = trainNetwork(X, y, δίκτυο, varargin)

Ορίζει τη συνάρτηση «trainNetwork» με εισόδους «X», «y», «network» και προαιρετικές παραμέτρους «varargin». Η συνάρτηση θα επιστρέψει τα «trainedNetwork», «cost_log», «trainingSetAccuracy» και «validationSetAccuracy».

Προεπιλεγμένη προετοιμασία παραμέτρων:

```
defaultEpochs=100;
defaultAlpha=0,01;
defaultValidationData=[];
defaultValidationOutput=[];
```

Ορίζει τις προεπιλεγμένες τιμές για τον αριθμό των κύκλων εκπαίδευσης (εποχές), τον ρυθμό εκμάθησης («άλφα»), τα δεδομένα επικύρωσης και τις ετικέτες επικύρωσης.

Ανάλυση εισόδου:

```
p = inputParser;
   p.FunctionName = 'trainNetwork';

addParameter(p,'epochs',defaultEpochs,@(x)validateattributes_with_return_value(x,{'n
umeric'},{'nonempty'}));

addParameter(p,'alpha',defaultAlpha,@(x)validateattributes_with_return_value(x,{'num
eric'},{'nonempty'}));
   addParameter(p,'validationData',defaultValidationData);
   addParameter(p,'validationDataOutput',defaultValidationOutput);
   p.parse(varargin{:});
```

Ρυθμίζει ένα αντικείμενο «inputParser» για να χειρίζεται προαιρετικές παραμέτρους εισαγωγής («varargin»). Ελέγχει και εκχωρεί προεπιλεγμένες τιμές εάν ορισμένες παράμετροι δεν παρέχονται από τον χρήστη.

Εκχώρηση αναλυόμενων εισόδων:

```
epochs = p.Results.epochs;
   alpha = p.Results.alpha;
   validationData = p.Results.validationData;
   validationOutput = p.Results.validationDataOutput;
```

Αντιστοιχίζει τα αναλυμένα αποτελέσματα στις αντίστοιχες μεταβλητές.

Έλεγχος επικύρωσης:

```
doValidation = ~isempty(validationData) && ~isempty(validationOutput);
```

Ελέγχει εάν παρέχονται δεδομένα επικύρωσης και ετικέτες και ορίζει τη σημαία «doValidation» ανάλογα.

Ενεργοποίηση δικτύου:

```
theta = network;
   numberOfThetas = length(theta);
   numberOfLayers = numberOfThetas +1;

   layer=cell(1,numberOfLayers);

   m=size(X,1);
   cost_log=zeros(epochs,1);
```

Αρχικοποιεί διάφορες μεταβλητές, συμπεριλαμβανομένων των βαρών («θήτα»), του αριθμού των παραδειγμάτων εκπαίδευσης («m») και της αποθήκευσης για τις τιμές κόστους και ακρίβειας ανά εποχές.

7. Εκχώρηση μεταφερθέντων δεδομένων εισόδου:

```
 layer{1} = X';
   layer{1}=[layer{1}; ones(1,size(layer{1},2))];
```

Αποθηκεύει τα μεταφερόμενα δεδομένα εκπαίδευσης στο πρώτο επίπεδο και προσθέτει μια μετατόπιση (bias) σε αυτό.

8. Βρόχος κατάβασης κλίσης:

```
      for i=1:epochs
      fprintf("Epoch %d/%d\r",i,epochs);
```

Αυτός είναι ο κύριος βρόχος όπου γίνεται η εκπαίδευση. Θα επαναληφθεί για τον αριθμό των «εποχών».

Μπροστινή διάδοση:

```
for j=1:numberOfThetas
        layer{j}(end,:)=1;
        layer{j+1} = sigmoid(theta{j} * layer{j});
end
```

Υπολογίζει τις εξόδους των επιπέδων δικτύου χρησιμοποιώντας τα τρέχοντα βάρη και τη συνάρτηση ενεργοποίησης σιγμοειδούς.

10. Πίσω διάδοση:

```
error=cell(1,numberOfLayers);
      error{numberOfLayers} = layer{numberOfLayers} - y';
      for j=numberOfThetas: -1 :2
         error{j} = theta{j}' * error {j+1};
      end
```

Υπολογίζει το σφάλμα για κάθε νευρώνα στο δίκτυο διαδίδοντας τα σφάλματα προς τα πίσω.

11. **Ενημέρωση βαρών**:

```
for j=1:numberOfThetas
        theta{j} = theta{j} - alpha * ((error{j+1} .* layer{j+1} .* (1-layer{j+1})) *
layer{j}');
 end
```

Ενημερώνει τα βάρη δικτύου («θ») με βάση τα υπολογισμένα σφάλματα.

12. Υπολογισμός κόστους και ακρίβειας:

```
cost = 1/m * sum(sum(error{numberOfLayers}.^2));
      cost_log(i)=cost;

      trainingSetAccuracy(i)=calculateAccuracy(layer{numberOfLayers}, y');

      if(doValidation)
         prediction=networkPrediction(validationData, theta);
         validationSetAccuracy(i)=calculateAccuracy(prediction, validationOutput');
      end

   end
```

Υπολογίζει και αποθηκεύει το μέσο τετραγωνικό σφάλμα, την ακρίβεια εκπαίδευσης και, εάν ισχύει, την ακρίβεια επικύρωσης για την τρέχουσα εποχή.

14. Τελικό δίκτυο επιστροφής:

```
trainedNetwork = theta;
```

Εκχωρεί τα εκπαιδευμένα βάρη στη μεταβλητή επιστροφής `trainedNetwork`.

Αυτή είναι μια ανάλυση γραμμή προς γραμμή. Ουσιαστικά είναι ένας αλγόριθμος εκπαίδευσης κατάβασης κλίσης για ένα νευρωνικό δίκτυο, με υποστήριξη για παρακολούθηση κόστους και ακρίβειας σε εποχές, καθώς και προαιρετική επικύρωση.

Ακολουθεί μια ανάλυση του σεναρίου OCTAVE, το οποίο εστιάζει στην εκπαίδευση ενός νευρωνικού δικτύου για ταξινόμηση εικόνων και στη χρήση του εκπαιδευμένου δικτύου για πρόβλεψη εικόνας:

1. Αρχικοποίηση και ορισμός σταθερών

```
clear; clc; close all;

fprintf('Starting Script \n')
epochs = 400;
alpha = 0.00001;

width=1080;
height=675;

color1=[255;255;255];
color0=[0;0;0];

rand ("seed", 123456);
```

Διαγράφει τον χώρο εργασίας, το παράθυρο εντολών και κλείνει όλα τα παράθυρα σχημάτων.
Καθορίζει σταθερές που σχετίζονται με εποχές εκπαίδευσης, ρυθμό μάθησης και διαστάσεις και χρώματα εικόνας.

2. Προετοιμασία δεδομένων εισαγωγής

```
fprintf('Reading and Preparing Training Data \n')
inputPicture = imread('training2.png');
labelPicture = imread('labels_color2.png');
inputPicture = cast(inputPicture,'double'); %Need to be casted from uint to double
labels = zeros(height,width) + 0.01;
labels(labelPicture(:,:,2)>0)=0.99;
labels = reshape(labels,[],1);
inputPicture = reshape(inputPicture,[],3);
numCategoryOne=(sum(labels==0.99)*100)/(width*height);
fprintf('Statistics:\n');
fprintf(' - Category 1: %2.2f %%\n',numCategoryOne);
fprintf(' - Background: %2.2f %%\n',100-numCategoryOne);
inputPicture = inputPicture/255;
```

Φορτώνει μια εικόνα και τις αντίστοιχες ετικέτες για εκπαίδευση.
Προ επεξεργάζεται δεδομένα εικόνας, μετατρέποντάς τα σε διπλή ακρίβεια και αναδιαμορφώνοντας.
Προετοιμασία του πίνακα με τις ετικέτες.
Οι είσοδοι κλιμακώνονται από 0 έως 255 σε μεταξύ 0 και 1 για χρήση της λειτουργίας ενεργοποίησης σιγμοειδούς.
Εμφανίζει το ποσοστό κάθε κατηγορίας στην εικόνα.

3. Δημιουργία Δικτύου

```
fprintf('Generate Network \n')
networkStructure = [3 3 1];
network = generateNetwork(networkStructure);
```

Καθορίζει τη δομή του νευρωνικού δικτύου.
Καλεί μια συνάρτηση «generateNetwork» για την προετοιμασία του δικτύου, η οποία αναλύθηκε προηγούμενως.

4. Εκπαιδευτική Διαδικασία

```
fprintf('Start Training \n')

[trainedNetwork,costLog,accuracyLog]=trainNetwork(inputPicture,labels,network,'epo
chs',epochs, 'alpha',alpha);

figCostLog=figure();
plot(costLog);
ylabel('loss');
xlabel('epochs');

fprintf('Training Done\n')
```

Καλεί τη συνάρτηση «trainNetwork» (από το προηγούμενο απόσπασμα κώδικα) για να εκπαιδεύσει το δίκτυο χρησιμοποιώντας τα προετοιμασμένα δεδομένα.
Σχεδιάζει το γράφημα απώλειας προπόνησης έναντι εποχών.

5. Πρόβλεψη

```
fprintf('Using Trained Network for Test Prediction\n')

for i=1:height
   for j=1:width
      if(predOutput(i,j)==1)
         predictionPicture(i,j,:)=color1;
      else
         predictionPicture(i,j,:)=color0;
      end
   end
end

predictionPicture = cast(predictionPicture,'uint8');

figure();
imshow(predictionPicture);
```

Χρησιμοποιεί το εκπαιδευμένο δίκτυο για να προβλέψει την ταξινόμηση της αρχικής εικόνας εισόδου.
Επεξεργάζεται εκ των υστέρων τα προβλεπόμενα αποτελέσματα, δημιουργώντας μια νέα εικόνα εκχωρώντας χρώματα με βάση τις προβλεπόμενες ετικέτες.

6. Δημιουργία και αποθήκευση αποτελεσμάτων

```
fprintf('Results \n')

nnParams = trainedNetwork;
nnParams{1} = nnParams{1}(1:end-1,:); %Ignore Last Column

fprintf('\nWeight Matrix from the Input to the Hidden Layer\n')
disp(nnParams{1});
fprintf('Weight Matrix from the Hidden to the Output Layer\n')
disp(nnParams{2});

save('NN_RGB_2_Categories_config.mat','trainedNetwork','networkStructure','nnPara
ms');

fprintf('\nFinished Script\n')
```

Εμφανίζει τους πίνακες βάρους από το εκπαιδευμένο νευρωνικό δίκτυο.
Αποθηκεύει τις διαμορφώσεις δικτύου και τους πίνακες βάρους σε ένα αρχείο «.mat».
Αυτό το σενάριο παρέχει μια ολοκληρωμένη ροή εργασιών από την προετοιμασία δεδομένων, την εκπαίδευση νευρωνικών δικτύων έως τη χρήση του εκπαιδευμένου δικτύου για προβλέψεις, παρέχοντας μια βάση για τη δημιουργία συστημάτων ταξινόμησης εικόνων με νευρωνικά δίκτυα.

Τέλος ακολουθεί η αναάλυση του κώδικα σε Python για την εκπαίδευση του NN και την χρήση της βιβλιοθήκης HLS4ML

1. Εισαγωγή βιβλιοθηκών

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

np.random.seed(2)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import itertools

from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

sns.set(style='white', context='notebook', palette='deep')
```

Αυτή η ενότητα εισάγει τις απαραίτητες βιβλιοθήκες και ρυθμίζει το περιβάλλον:

- pandas, numpy: Χειρισμός δεδομένων.
- matplotlib, seaborn: Οπτικοποίηση δεδομένων.
- %matplotlib inline: εμφάνιση γραφικών ενσωματωμένων.
- np.random.seed(2): Εξασφαλίζει την αναπαραγωγιμότητα ορίζοντας έναν σπόρο για τυχαία παραγωγή αριθμών.
- sklearn: Χρησιμοποιείται για διαχωρισμό δεδομένων και αξιολόγηση μοντέλων.
- keras: Βιβλιοθήκη Deep Learning, που χρησιμοποιείται για κατασκευή μοντέλων και εκπαίδευση.
- sns.set(style='white', context='notebook', palette='deep'): ορίζονται αισθητικές παραμέτρους για τα οικόπεδα.

2. Φόρτωμα δεδομένων και βασική ανάλυση

```
train = pd.read_csv("/home/nikos/Downloads/sdga/train.csv")
test = pd.read_csv("/home/nikos/Downloads/sdga/test.csv")

Y_train = train["label"]
X_train = train.drop(labels = ["label"],axis = 1)
del train

g = sns.countplot(Y_train)
Y_train.value_counts()
X_train.isnull().any().describe()
test.isnull().any().describe()
```

- pd.read_csv: Φόρτωση συνόλων δεδομένων.
- Y_train και X_train: Εξάγωνται ετικέτες και χαρακτηριστικά από τα δεδομένα εκπαίδευσης.
- del train: Απελευθερώνει τη μνήμη διαγράφοντας την πλέον περιττή μεταβλητή.
- sns.countplot: Οπτικοποιεί τη διανομή ετικετών.
- .isnull().any().describe(): Ελέγχει για μηδενικές τιμές στα σετ εκπαίδευσης και δοκιμής.

3. Προεπεξεργασία δεδομένων

```
X_train = X_train / 255.0
test = test / 255.0
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
Y_train = to_categorical(Y_train, num_classes = 10)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1,
random_state=2)
```

- / 255.0: Κανονικοποίηση τιμών εικονοστοιχείων στο εύρος [0, 1].
- .values.reshape(-1,28,28,1): Αλλαγή στο σχήμα των επίπεδων εικονοστοιχείων σε μορφή 28x28x1 (μορφή εικόνας).
- to_categorical: Μετατροπή ετικετών σε one-hot κωδικοποίηση.
- train_test_split: Διαχωρισμός δεδομένων σε σύνολα εκπαίδευσης και επικύρωσης.

4. Δημιουργία και Σύνταξη Μοντέλου CNN

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
        activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
        activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
          activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
          activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))

optimizer = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = optimizer , loss = "categorical_crossentropy",
metrics=["accuracy"])
```

Αυτή η ενότητα ορίζει το μοντέλο CNN επίπεδο προς επίπεδο:

Conv2D: Συνελικτικά επίπεδα για εξαγωγή χαρακτηριστικών.
MaxPool2D: Συγκέντρωση επιπέδων για μείωση δειγματοληψίας.
Dropout: Τεχνική τακτοποίησης για την αποφυγή υπερβολικής προσαρμογής.
Flatten: Μετατρέπει τα δεδομένα μήτρας 2D σε διάνυσμα.
Dense: Πλήρως συνδεδεμένα στρώματα για ταξινόμηση.
model.compile: Ρυθμίζει το μοντέλο για εκπαίδευση.

5. Ρυθμίστε την επιστροφή κλήσης μείωσης ρυθμού εκμάθησης

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

epochs = 1
```

ReduceLROnPlateau: Μια επανάκληση για τη μείωση του ρυθμού εκμάθησης όταν η ακρίβεια επικύρωσης πλησιάζει.

6. Επαύξηση Δεδομένων

```
datagen = ImageDataGenerator(
      featurewise_center=False,  # set input mean to 0 over the dataset
      samplewise_center=False,  # set each sample mean to 0
      featurewise_std_normalization=False,  # divide inputs by std of the dataset
      samplewise_std_normalization=False,  # divide each input by its std
      zca_whitening=False,  # apply ZCA whitening
      rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
      zoom_range = 0.1, # Randomly zoom image
      width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
```

```
width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total
height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

ImageDataGenerator: Δημιουργεί επαυξημένες εικόνες on-the-fly κατά τη διάρκεια της προπόνησης.
datagen.fit: Προσαρμογή του augmentor στα δεδομένα.

7. Εκπαίδευση μοντέλου

```
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
                              , callbacks=[learning_rate_reduction])
```

model.fit: Εκπαίδευση του μοντέλου με καθορισμένα δεδομένα, εποχές και επανακλήσεις (callbacks).
datagen.flow: Χρησιμοποιεί επαυξημένα δεδομένα στην εκπαίδευση.

8. Οπτικοποίηση, Πρόβλεψη

```
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))

# Display some error results

Y_pred_classes_errors = Y_pred_classes[errors]
Y_pred_errors = Y_pred[errors]
Y_true_errors = Y_true[errors]
X_val_errors = X_val[errors]

def display_errors(errors_index,img_errors,pred_errors, obs_errors):
    n = 0
    nrows = 2
    ncols = 3
    fig, ax = plt.subplots(nrows,ncols,sharex=True,sharey=True)
    for row in range(nrows):
        for col in range(ncols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((28,28)))
            ax[row,col].set_title("Predicted label :{}\nTrue label
:{}".format(pred_errors[error],obs_errors[error]))
            n += 1

# Probabilities of the wrong predicted numbers
Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)

# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))

# Difference between the probability of the predicted label and the true label
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors

# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)
```

```
# Top 6 errors
most_important_errors = sorted_dela_errors[-6:]

# Show the top 6 errors
display_errors(most_important_errors, X_val_errors, Y_pred_classes_errors,
Y_true_errors)


# predict results
results = model.predict(test)

# select the indix with the maximum probability
results = np.argmax(results,axis = 1)

results = pd.Series(results,name="Label")
```

Αυτό το κομμάτι του κώδικα στην ουσία οπτικοποιεί κάποιες λάθος ταξινομήσεις και εμφανίζει την εικόνα του ψηφίου με δύο labels, την λάθος πρόβλεψη και την σωστή αντιστοίχιση.

9. Μετατροπή μοντέλου για FPGA

```
import plotting
import hls4ml
config = hls4ml.utils.config_from_keras_model(model, granularity='model')
print("-----------------------------------")
print("Configuration")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                    hls_config=config,
                                    output_dir='model_nikos/hls4ml_prj',
                                    part='xcu250-figd2104-2L-e')

hls4ml.utils.plot_model(hls_model, show_shapes=True, show_precision=True,
    to_file=None)

    hls_model.compile()
```

Τέλος εφαρμόζεται η βιβλιοθήκη HLS4ML για την μετατροπή του μοντέλου για την πλακέτα FPGA.

REFERENCES

[1] Real-Time Machine Vision FPGA Implementation for Microfluidic Monitoring on Lab-on-Chips, L. Sotiropoulou, L. Voudouris, C. Gentsos, A. M. Demiris, N. Vassiliadis and S. Nikolaidis,
https://ieeexplore.ieee.org/abstract/document/6544300/authors#authors

[2] NPE: An FPGA-based Overlay Processor for Natural Language Processing
Hamza Khan, Asma Khan, Zainab Khan, Lun Bin Huang, Kun Wang, Lei He
https://arxiv.org/abs/2104.06535

[3] Real-Time Formal Verification of Autonomous Systems With An FPGA
Minh Bui, Michael Lu, Reza Hojabr, Mo Chen, Arrvindh Shriraman
https://arxiv.org/abs/2012.04011

[4] Τεχνητή Νοημοσύνη - Β' Έκδοση,
Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, Η. Σακελλαρίου

[5] G Field-programmable gate array Wikipedia, https://en.wikipedia.org/wiki/Field-programmable_gate_array#History

[6] Real World Multicore Embedded Systems,
Bryon Moyer, Yosinori Watanabe

[7] FPGA Implementation of a Deep Learning Acceleration Core Architecture for Image Target Detection,
Xu Yang, Chen Zhuang, Wenquan Feng, Zhe Yang, Qiang Wang

[8] Internet of Things and Cyber-Physical Systems,
Raghubir Singh, Sukhpal Singh Gill

[9] High-performance computing using FPGAs,
Wim Vanderbauwhede, Khaled Benkrid

[10] FPGA Security: Motivations, Features, and Applications,
Stephen Trimberger, Jason J. Moore

[11] Acceleration of Trading System Back End with FPGAs Using High-Level Synthesis Flow,
Sunil Puranik, Mahesh Barve, Swapnil Rodi, Rajendra Patrikar

[12] Hardware Architectures for Real-Time Medical Imaging,
Alcaín, E., Fernández, P. R., Nieto, R., Montemayor, A. S., Vilas, J., Galiana-Bordera, A., Martinez-Girones, P. M., Prieto-de-la-Lastra, C., Rodriguez-Vila, B., Bonet, M., Rodriguez-Sanchez, C., Yahyaoui, I., Malpica, N., Borromeo, S., Machado, F., & Torrado-Carvajal, A.

[13] A Soft Coprocessor Approach for Developing Image and Video Processing Applications on FPGAs,

[14] FPGA-Based Low-Power Speech Recognition with Recurrent Neural Networks,

Minjae Lee, Seoul National University, Kyuyeon Hwang, Jinhwan Park,
Sungwook Choi

[15] Robotic Computing on FPGAs: Current Progress, Research Challenges, and Opportunities,
Zishen Wan, Ashwin Lele, Bo Yu, Shaoshan Liu, Yu Wang, Vijay Janapa Reddi, Cong Hao, Arijit Raychowdhury

[16] State of the Art FPGAs Design Optimizations, Amit Singh Bhatti
https://amit02093.medium.com/state-of-the-art-fpgas-design-optimizations-15e9778e60af

[17] Towards Machine Learning-Based FPGA Backend Flow: Challenges and Opportunities, Taj I, Farooq U
https://www.mdpi.com/2079-9292/12/4/935

[18] FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?
Anouar Nechi, Lukas Groth, Saleh Mulhem, Farhad Merchant, Rainer Buchty, Mladen Berekovic
https://dl.acm.org/doi/10.1145/3613963

[19] Machine Learning for the Control and Monitoring of Electric Machine Drives: Advances and Trends, Shen Zhang, Oliver Wallscheid, Mario Porrmann
https://ieeexplore.ieee.org/document/10147346

[20] What is gradient descent?, IBM
https://www.ibm.com/topics/gradient-descent

[21] Gradient Descent Algorithm — a deep dive, Robert Kwiatkowski
https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

[22] FPGA Vision Remote Lab, Marco Winzker
https://www.h-brs.de/de/fpga-vision-lab