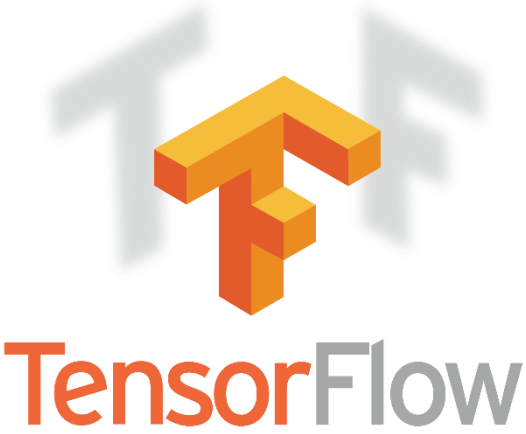
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

<<Κατανεμημένη εκπαίδευση μοντέλων Μηχανικής Μάθησης>>



Των φοιτητών

Μουμουλίδη Αλέξανδρου (ΑΜ: 113755)

Τζούμα Χρήστου (ΑΜ: 123971)

Επιβλέπων καθηγητής

Κ.Διαμαντάρας Κων/νος

Θεσσαλονίκη 2018

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία αποσκοπεί στην παρουσίαση πλατφορμών, εργαλείων και τεχνικών που αφορούν την εκπαίδευση κατανεμημένων μοντέλων μηχανικής μάθησης μέσω του Tensorflow της Google και σύγκριση με τους ανταγωνιστές, και τους πόρους τους οποίους παρέχουν για να διευκολήνουν τους μηχανικούς της τεχνολογίας δεδομένων. Επιπρόσθετα αναλύονται οι όροι Μηχανική Μάθηση και Κατανεμημένη Εκπαίδευση και ο ρόλος τους στις βιομηχανίες του σήμερα. Διερευνώνται επίσης οι τρόποι και οι τεχνικές εξαγωγής ενός εκπαιδευμένου μοντέλου μηχανικής μάθησης, για την χρήση και εφαρμογή του στις Mobile πλατφόρμες Android και iOS. Στην συνέχεια παρουσιάζεται η εκπαίδευση ενός κατανεμημένου μοντέλου σε Tensorflow, μέσω της γλώσσας Python και των βιβλιοθηκών της. Το μοντέλο αυτό εκπαιδεύεται πάνω στο MNIST dataset με στόχο την δημιουργία ενός συστήματος που μπορεί να αναγνωρίσει αριθμητικά ψηφία απο το 0 εως το 9, δίνοντας του σαν είσοδο μια εικόνα μεγέθους 28x28 σε κλίμακα του γκρι. Κατόπιν αυτό το μοντέλο το επεξεργαζόμαστε και το μετατρέπουμε στην μορφή που είναι απαραίτητη για την χρήση του, απο τα ανάλογα frameworks σε Android και σε iOS. Τέλος δημιουργούμε τις ανάλογες εφαρμογές οι οποίες επιτρέπουν σε ένα χρήστη να ζωγραφίσει ένα αριθμητικό ψηφίο απο το 0 εώς το 9 και πατώντας ένα κουμπί να του εμφανίσει την πρόβλεψη του μοντέλου που εισήχθη. Η εργασία εκπονήθηκε από τους Μουμουλίδη Αλέξανδρο και Τζούμα Χρήστο, φοιτητές του Αλεξάνδρειου Τεχνολογικού Ιδρύματος Θεσσαλονίκης με επιβλέπων καθηγητή τον κ. Διαμαντάρα Κωνσταντίνο.

Λέξεις κλειδιά: Tensorflow, Tensorflow Lite, Distributed Training, MLAAS, Cloud Computing, Softmax Regression, Deep Learning, Big Data

ABSTRACT

This thesis aims at presenting platforms, tools and techniques for the training of distributed Machine Learning models through Google's Tensorflow and comparing it with the competitors of this platform and the resources they provide to facilitate datascience technology engineers and programmers. In addition, the terms Machine Learning and Distributed Training are analyzed for their role in today's industries. We also explore the ways and techniques of exporting a trained model of learning mechanics to use and implement it on Mobile Platforms on Android and iOS. Then we showcase the training of a distributed model in Tensorflow, through Python and its libraries. This model is trained on the MNIST dataset to create a system that can recognize numeric digits from 0 to 9, by taking it as an input of a greyscale 28x28 image. We then process this model and convert it into the form it needs to be in order to be utilised, from the corresponding frameworks of Android and iOS. Finally, we create the applications that allow a user to draw a numeric digit from 0 to 9 and by pressing a button the screen displays the predicted outcome of the model we previously imported. The thesis was conducted by Moumoulides Alexandros and Tzoumas Christos, students of Alexander Technological Institute of Thessaloniki and was supervised by professor Mr. Diamantaras Konstantinos.

Keywords: Tensorflow, Tensorflow Lite, Distributed Training, MLAAS, Cloud Computing, Softmax Regression, Deep Learning, Big Data

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία, εκπονήθηκε κατά το ακαδημαϊκό έτος 2017-2018 για το Τμήμα Μηχανικών Πληροφορικής του Αλεξάνδρειου Τεχνολογικού Εκπαιδευτικού Ιδρύματος.

Επιβλέπων καθηγητής της εργασίας, ήταν ο καθηγητής κ. Διαμαντάρας Κωνσταντίνος, τον οποίο ευχαριστούμε για την ανάθεση του θέματος, την εμπιστοσύνη και την καθοδήγησή του καθ' όλη την διάρκεια εκπόνησης. Πρόκειται για ένα θέμα με το οποίο δεν είχαμε σχέση στο παρελθόν, όμως διεύρυνε ιδιαίτερα τις γνώσεις μας και το ενδιαφέρον μας για περαιτέρω και βαθύτερη ενασχόληση με αυτό.

Επίσης, θέλουμε να αναφέρουμε την μεγάλη βοήθεια που πήραμε από μεγάλες κοινότητες υποστήριξης στο διαδίκτυο, βρίσκοντας απαντήσεις σε προβλήματα που προέκυπταν για εμάς, είχαν όμως ήδη λυθεί σε αντίστοιχες περιπτώσεις από άλλους μηχανικούς πληροφορικής.

Τέλος, θα θέλαμε να ευχαριστήσουμε πολύ τις οικογένειες και τους οικείους μας, οι οποίοι έδειξαν κατανόηση και μας υποστήριξαν με όποιον δυνατό τρόπο να ολοκληρώσουμε αυτή την εργασία στα χρονικά πλαίσια που συμφωνήθηκαν.

Μουμουλίδης Αλέξανδρος & Τζούμας Χρήστος

Ιούνιος 2018

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Σε γενικές γραμμές, η κατανεμημένη μηχανική μάθηση (Distributed Machine Learning) είναι ένας διεπιστημονικός τομέας που συνδυάζει σχεδόν κάθε τομέα της επιστήμης των υπολογιστών όπως στατιστική, θεωρία της μάθησης, βελτιστοποίηση, αλγόριθμοι, Machine learning (deep learning, γραφικά μοντέλα, κλπ), ακόμη και συστήματα διανομής και αποθήκευσης. Υπάρχουν αμέτρητα προβλήματα που μπορούν να διερευνηθούν και να μελετηθούν σε καθέναν από αυτούς τους υποτομείς. Επίσης είναι η περισσότερο υιοθετημένη και αναπτυχθείσα τεχνολογία της Μηχανικής Μαθησης στην βιομηχανία, λόγο της ικανότητάς της να χειρίζεται, να αναλύει και να παράγει μεγάλες ποσότητες δεδομένων (Big Data). Καθώς τα σύνολα των δεδομένων αυξάνονται σε μέγεθος και τα νευρωνικά δίκτυα σε πολυπλοκότητα, αναλογικά αυξάνονται και οι απαιτήσεις της μνήμης και της υπολογιστικής δύναμης. Η εκπαίδευση ενός DNN (Deep Neural Network - Βαθύ Νευρωνικό Δίκτυο) σήμερα μπορεί να επιτευχθεί σε ένα υψηλό βαθμό ακρίβειας μόνο με την χρήση ένος συμπλέγματος μηχανών (Cluster) με υπολογιστική αρχιτεκτονική υψηλής απόδοσης. Για την βέλτιστη αξιοποίηση της υπολογιστική ισχύος που διατίθεται σε αυτά τα συστήματα, διάφορες τεχνικές εκπαίδευσης και συμπεράσματος (inference) των DNN έχουν επινοηθεί και τροποποιηθεί για να αυξήσουν την αποτελεσματικότητα και την επικοινωνία μεταξύ τους. Σε αυτή την εργασία καταγράφονται κάποιες από αυτές τις τεχνικές και κάποιες πλατφόρμες και εργαλεία που διευκολύνουν την χρήση τέτοιων συστημάτων.

ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Γέννηση [1952 - 1956]
1950 - Ο Alan Turing δημιουργεί το "Test Turing" για να διαπιστώσει αν μια μηχανή είναι πραγματικά έξυπνη. Για να περάσει τη δοκιμή, το μηχάνημα πρέπει να είναι ικανό να κάνει έναν άνθρωπο να πιστεύει ότι είναι ένας άλλος άνθρωπος αντί ενός υπολογιστή.

1952 - Ο Arthur Samuel γράφει το πρώτο πρόγραμμα υπολογιστή ικανό να μάθει. Το λογισμικό ήταν ένα πρόγραμμα που θα μπορούσε να παίξει πούλια(checkers) και να βελτιωθεί με κάθε παιχνίδι που έπαιζε.

1956 - Ο Martin Minsky και ο John McCarthy, με τη βοήθεια του Claude Shannon και του Nathan Rochester, πραγματοποίησαν μια διάσκεψη στο Dartmouth το 1956, το οποίο θεωρείται ότι είναι το σημείο όπου γεννήθηκε το πεδίο της Τεχνητής Νοημοσύνης. Ο Μίνσκι έπεισε τους παρευρισκόμενους να υιοθετήσουν τον όρο "Τεχνητή Νοημοσύνη" ως το όνομα για το νέο πεδίο.

1958 - Ο Frank Rosenblatt σχεδιάζει το Perceptron, το πρώτο τεχνητό νευρωνικό δίκτυο.

1967 - Γράφετε ο αλγόριθμος "Nearest Neighbor". Αυτό το ορόσημο θεωρείται η γέννηση του πεδίου της αναγνώρισης προτύπων στους υπολογιστές.

Πρώτος Χειμώνας του AI [1974 - 1980]
Το δεύτερο μισό της δεκαετίας του 1970, ο τομέας υπέστη τον πρώτο «χειμώνα». Διάφορα ινστιτούτα και επιχχειρήσεις που χρηματοδοτούσαν την έρευνα της τεχνητής νοημοσύνης έκοψαν τα κεφάλαια μετά από χρόνια με μεγάλες προσδοκίες και μικρή πραγματική πρόοδο.

1979 - Οι φοιτητές του Πανεπιστημίου του Στάνφορντ επινοούν το "Stanford Cart", ένα κινητό ρομπότ ικανό να κινείται αυτόνομα γύρω από μια αίθουσα, αποφεύγοντας τα εμπόδια.

Η Έκρηξη της δεκαετίας του 1980 [1980 - 1987]
Η δεκαετία του '80 είναι γνωστή για την γέννηση των “expert systems” ειδικών συστημάτων, βασισμένων σε κανόνες. Αυτά υιοθετήθηκαν γρήγορα από τον εταιρικό τομέα, δημιουργώντας νέο ενδιαφέρον για τη Μηχανική Μάθηση.

1981 - Ο Gerald Dejong εισάγει την έννοια της “Explanation Based Learning”(Εκμάθησης βασισμένης στη μάθηση), στην οποία ένας υπολογιστής αναλύει τα δεδομένα εκπαίδευσης και δημιουργεί γενικούς κανόνες που επιτρέπουν την απόρριψη των λιγότερο σημαντικών δεδομένων.

1985 - Ο Terry Sejnowski εφευρίσκει το NetTalk, το οποίο μαθαίνει να προφέρει λέξεις με τον ίδιο τρόπο που μαθαίνει ένα παιδί.

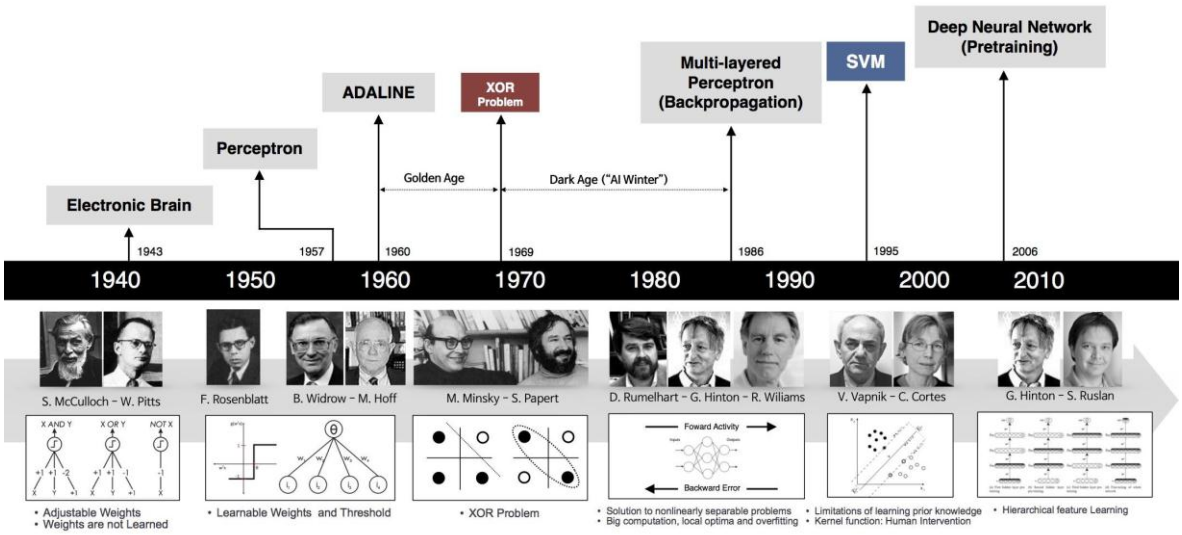
Δεύτερος AI Winter [1987 - 1993]
Στα τέλη της δεκαετίας του '80 και στις αρχές της δεκαετίας του '90, ο τομέας του AI γνώρισε ένα δεύτερο "χειμώνα". Αυτή τη φορά, τα αποτελέσματά του διήρκεσαν για αρκετά χρόνια και η φήμη του τομέα δεν ανακτήθηκε πλήρως μέχρι τις αρχές της δεκαετίας του 2000.

Η δεκαετία του '90 - Η τεχνολογία της μηχανικής μάθησης μετακινείται από την εστίαση στη ανάπτυξη μοντέλων βασισμένα στην γνώση σε μοντέλα που βασίζονται στην μάθηση μέσω μεγάλων ποσοτήτων δεδομένων. Οι επιστήμονες αρχίζουν να δημιουργούν προγράμματα που αναλύουν μεγάλες ποσότητες δεδομένων και εξαγάγουν συμπεράσματα από τα αποτελέσματα.

1997 - Ο υπολογιστής Deep Blue της IBM, νικά τον παγκόσμιο πρωταθλητή σκακιού Gary Kasparov.

Έκρηξη και εμπορευματοποίηση [2006 - Σήμερα]
Η αύξηση της υπολογιστικής δύναμης των υπολογιστών μαζί με τη μεγάλη αφθονία των διαθέσιμων δεδομένων ξαναζωντάνεψαν το πεδίο της Μηχανικής Μάθησης. Πολλές επιχειρήσεις στρέφουν τις εταιρείες τους προς την συλλογή δεδομένων και ενσωματώνουν τη Μηχανική Μάθηση στις διαδικασίες, τα προϊόντα και τις υπηρεσίες τους, προκειμένου να αποκτήσουν πλεονέκτημα έναντι του ανταγωνισμού τους.

2006 - Ο Geoffrey Hinton εφήυρε τη φράση "Deep Learning" για να εξηγήσει τις νέες αρχιτεκτονικές των βαθιών νευρωνικών δικτύων ικανών να μάθουν πολύ καλύτερα μοντέλα.



Εικόνα 1: Ιστορική αναδρομή

2011 - Ο υπολογιστής Watson από την IBM νικά τους ανθρώπινους ανταγωνιστές στο Jeopardy, ένα τηλεοπτικό παιχνίδι που αποτελείται από απαντήσεις σε ερωτήσεις στη φυσική γλώσσα.

2012 - Ο Jeff Dean, με την βοήθεια του Andrew Ng (Πανεπιστήμιο του Στάνφορντ), ηγείται του GoogleBrain, το οποίο ανέπτυξε ένα βαθύ νευρωνικό δίκτυο χρησιμοποιώντας όλη την ικανότητα της υποδομής της Google για να ανιχνεύσει μοτίβα σε βίντεο και εικόνες.

2012 - Ο Geoffrey Hinton οδηγεί τη νικήτρια ομάδα στον διαγωνισμό Computer Vision στο Imagenet χρησιμοποιώντας ένα βαθύ νευρωνικό δίκτυο. Η ομάδα κέρδισε με μεγάλο περιθώριο, προκαλώντας την τρέχουσα έκρηξη της Μηχανικής Μάθησης με βάση τα Deep Neural Networks.

2012 - Το ερευνητικό εργαστήριο Google X χρησιμοποιεί το GoogleBrain για να αναλύει αυτόνομα τα βίντεο του Youtube και να ανιχνεύει αυτά που περιέχουν γάτες.

2014 - Το Facebook αναπτύσσει το DeepFace, έναν αλγόριθμο που βασίζεται σε DNN ικανά να αναγνωρίσουν ανθρώπους με την ίδια ακρίβεια με τον άνθρωπο.

2014 - Το Google αγοράζει το DeepMind, ένα βρετανικό startup που ειδικεύεται στο deep learning, το οποίο είχε πρόσφατα καταδείξει τις δυνατότητες του με έναν αλγόριθμο ικανό να παίζει παιχνίδια Atari απλά βλέποντας τα pixel στην οθόνη, όπως θα έκανε και ο άνθρωπος. Ο αλγόριθμος, μετά από ώρες εκπαίδευσης, ήταν ικανός να κερδίσει ανθρώπους εμπειρογνώμονες στα παιχνίδια.

2015 - Η Amazon εγκαινιάζει τη δική της πλατφόρμα εκμάθησης μηχανών.

2015 - Η Microsoft δημιουργεί το “Distributed Machine Learning Toolkit”, το οποίο επιτρέπει την αποτελεσματική κατανομή των προβλημάτων μηχανικής μάθησης σε πολλούς υπολογιστές.

2015 - Ο Elon Musk και ο Sam Altman, μεταξύ άλλων, ιδρύουν τον μη κερδοσκοπικό οργανισμό OpenAI, παρέχοντάς του ένα δισεκατομμύριο δολάρια με στόχο να εξασφαλίσει ότι η τεχνητή νοημοσύνη έχει θετικό αντίκτυπο στην ανθρωπότητα.

2016 - Το Google DeepMind με το μοντέλο AlphaGo νικά τον επαγγελματία παίχτη του παιχνιδιού “Go” Lee Sedol 5 - 1 , το οποίο παιχνίδι θεωρείται ένα από τα πιο πολύπλοκα επιτραπέζια παιχνίδια. Οι επαγγελματίες παίκτες “Go” επιβεβαίωσαν ότι ο αλγόριθμος ήταν σε θέση να κάνει "δημιουργικές" κινήσεις που δεν είχαν ξαναδεί.

2017 - Η εταιρία Waymo βγάζει στην παραγωγή τα πρώτα πραγματικά αυτόνομα αυτοκίνητα με πραγματικούς αναβάτες χωρίς να υπάρχει ανθρώπινος χειριστής στο τιμόνι.

Η ΕΝΝΟΙΑ ΤΗΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Η Μάθηση (Learning) είναι μία από τις θεμελιώδεις ιδιότητες της νοήμονος συμπεριφοράς του ανθρώπου. Είναι η διαδικασία κατά την οποία ο υποβαλλόμενος σε αυτήν αποκτά γνώσεις, δεξιότητες, συμπεριφορές και αξίες μέσα από την παρατήρηση του περιβάλλοντος, την σύγκριση και με την εφαρμογή γνωστικών διαδικασιών. Παρόλο που ο όρος μάθηση υποδηλώνει τη μαθησιακή *διαδικασία*, ωστόσο συχνά προσδιορίζει και το *αποτέλεσμα* αυτής. Οι επιστήμονες του χώρου της πληροφορικής και της Τεχνητής Νοημοσύνης (ΤΝ) ορίζουν την λεγόμενη Μηχανική Μάθηση (Machine Learning) ως:

το φαινόμενο κατά το οποίο ένα σύστημα βελτιώνει την απόδοσή του κατά την εκτέλεση μιας συγκεκριμένης εργασίας, χωρίς να υπάρχει ανάγκη να προγραμματιστεί εκ νέου

την δημιουργία, δηλαδή, υπολογιστικών συστημάτων που είναι ικανά να **μάθουν**, και να βελτιώσουν την απόδοσή τους κάνοντας κατάλληλη χρήση των αποτελεσμάτων κάποιας εργασίας.

Ένας σχετικός γενικός ορισμός Μηχανικής Μάθησης δίνεται από τον Mitchell (1997):

«Ένα πρόγραμμα υπολογιστή λέμε ότι μαθαίνει από την εμπειρία Ε ως προς κάποια κλάση εργασιών Τ και μέτρο απόδοσης Ρ, αν η απόδοσή του σε εργασίες από το Τ, όπως μετριέται από το Ρ, βελτιώνεται μέσω της εμπειρίας Ε.»

Στην Επαγωγική Μάθηση (Inductive Learning), με τη διαδικασία της επαγωγής (induction) ο άνθρωπος μαθαίνει κατανοώντας το περιβάλλον του μέσω παρατηρήσεων και δημιουργεί μια απλοποιημένη (αφαιρετική) εκδοχή του που ονομάζεται **νοητικό μοντέλο** (mental model). Επιπλέον, ο άνθρωπος έχει τη δυνατότητα να οργανώνει και να συσχετίζει τις εμπειρίες και τις παρατηρήσεις του δημιουργώντας νέες δομές που ονομάζονται νοητικά **πρότυπα** (mental patterns), με αξιοποίηση και του επαγωγικού και του απαγωγικού συλλογισμού.

Στη δημιουργία νέων προτύπων από παλαιά βασίζονται οι τρόποι μάθησης που εξαρτώνται σε μικρό ή μεγάλο βαθμό από την προϋπάρχουσα γνώση για ένα πρόβλημα, όπως είναι η μάθηση από επεξηγήσεις και η μάθηση από περιπτώσεις. Σε σχέση με την ανθρώπινη ικανότητα προς μάθηση, οι φιλόσοφοι θέτουν το ερώτημα: «*Πώς μπορεί ένας επαγωγικός συλλογισμός που οδηγεί στη μάθηση να αξιολογηθεί ως προς την ορθότητά του*;». Αντίστοιχα, οι ψυχολόγοι ρωτούν: «*Πώς*

αποθηκεύει ο εγκέφαλος τα αποτελέσματα της διαδικασίας της μάθησης, δηλαδή τα νοητικά μοντέλα και τα πρότυπα;». Στο χώρο της ΤΝ απλώς ρωτούν: «Πώς μπορεί μία μηχανή να δημιουργήσει νέα μοντέλα και πρότυπα μάθησης από συγκεκριμένα παραδείγματα και πόσο αξιόπιστα είναι αυτά τα μοντέλα και πρότυπα στην πράξη;». Με βάση τα παραπάνω, μπορεί να δοθεί ο ακόλουθος εναλλακτικός ορισμός για τη Μηχανική Μάθηση:

Η ικανότητα ενός υπολογιστικού συστήματος να δημιουργεί μοντέλα ή πρότυπα από ένα σύνολο δεδομένων.

Ως κλάδος της ΤΝ, η Μηχανική Μάθηση ασχολείται με τη μελέτη αλγορίθμων που βελτιώνουν τη συμπεριφορά τους σε κάποια εργασία που τους έχει ανατεθεί χρησιμοποιώντας την εμπειρία τους από προηγούμενες εκτελέσεις της ίδιας ή παραπλήσιας εργασίας σε διαφορετικά δεδομένα. Όσον αφορά τη σχεδίαση των συστημάτων Μηχανικής Μάθησης, για τα συστήματα που ανήκουν στη συμβολική ΤΝ, η δυνατότητα μάθησης προσδιορίζεται ως η ικανότητα πρόσκτησης επιπλέον γνώσης, που επιφέρει μεταβολές στην υπάρχουσα καταχωρημένη γνώση είτε αλλάζοντας χαρακτηριστικά της είτε με αυξομείωσή της. Στην περίπτωση των συστημάτων ΤΝ που ανήκουν στη Μη Συμβολική ΤΝ (όπως η περίπτωση των Τεχνητών Νευρωνικών Δικτύων), ως μάθηση προσδιορίζεται η δυνατότητα που διαθέτουν τα συστήματα στο να μετασχηματίζουν την εσωτερική τους δομή, παρά στο να μεταβάλλουν κατάλληλα τη γνώση που έχει καταχωρηθεί μέσα σε αυτά κατά το σχεδιασμό τους. Αν και απέχουμε πάρα πολύ από τη δημιουργία μηχανών που μαθαίνουν τόσο καλά όσο ο άνθρωπος, για συγκεκριμένες περιοχές μάθησης έχουν αναπτυχθεί αλγόριθμοι οι οποίοι έχουν επιτρέψει την εμφάνιση σύγχρονων εμπορικών εφαρμογών με σημαντική επιτυχία. Επιπλέον, τα αποτελέσματα από τις εφαρμογές της ΤΝ αρχίζουν ήδη να είναι ορατά και να δίνουν απαντήσεις σε αναπάντητα, έως τώρα, ερωτήματα άλλων κλάδων που διερευνούν την ικανότητα του ανθρώπου να μαθαίνει. Εκτός της ίδιας της ΤΝ, μεταξύ των επιστημονικών κλάδων που επωφελούνται από τα επιτεύγματα στον τομέα της Μηχανικής Μάθησης συγκαταλέγονται οι: Εξόρυξη Δεδομένων, Πιθανότητες και Στατιστική, Θεωρία της Πληροφορίας, Αριθμητική Βελτιστοποίηση, Θεωρία της Πολυπλοκότητας, Θεωρία Ελέγχου (προσαρμοστική), Ψυχολογία (εξελικτική, γνωστική), Νευροβιολογία και Γλωσσολογία.

1.1 ΚΑΤΑΝΕΜΗΜΕΝΗ ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Όταν δημιουργούμε συστήματα για την επίλυση πολύ μεγάλων προβλημάτων μηχανικής μάθησης, αντιμετωπίζουμε συνήθως το εξής πρόβλημα: Οι εργασίες πρέπει να μοιραστούν σε πολλά μηχανήματα, τα οποία να μοιράζονται πληροφορίες για την συνολική κατάσταση, δηλαδή παραμέτρους, πληροφορίες του συνολικού συστήματος (cluster), προφίλ χρηστών ή οτιδήποτε άλλο χρειάζεται να ρέει σαν πληροφορία μεταξύ τους. Η αύξηση του μεγέθους δεδομένων εισόδου για πολλούς αλγορίθμους εκπαίδευσης μπορεί να μειώσει σημαντικά το ποσοστό σφάλματος του μοντέλου. **Το Distributed Machine Learning** (κατανεμημένη μηχανική μάθηση) αναφέρεται σε αλγόριθμους μάθησης πολλαπλών κόμβων (συσκευών) και συστημάτων που έχουν σχεδιαστεί για τη βελτίωση της απόδοσης, την αύξηση της ακρίβειας και την κλιμάκωση τον δεδομένων εισόδου ενός μοντέλου μηχανικής μάθησης. Η κατανεμημένη εκμάθηση μηχανών επιτρέπει σε εταιρείες, ερευνητές και ιδιώτες να λαμβάνουν τεκμηριωμένες αποφάσεις και να αντλούν σημαντικά συμπεράσματα από μεγάλα ποσά δεδομένων.

Σε ρεαλιστικές εφαρμογές του πραγματικού κόσμου οι οποίες χρησιμοποιούν την μηχανική μάθηση για να παράγουν αποτελέσματα, τα σύνολα των δεδομένων που πρέπει να επεξεργαστούν συνήθως κυμαίνονται από 1TB(Τεραμπάϊτ) έως 1PB(Πεταμπάϊτ). Για παράδειγμα αν ένα κοινωνικό δίκτυο με 100 εκατομμύρια χρήστες κρατάει 1KB(Κιλομπάϊτ) δεδομένων ανά χρήστη τότε θα πρέπει να έχει την δυνατότητα να επεξεργαστεί 100TB από δεδομένα χρηστών. Τέτοιες τεράστιες ποσότητες δεδομένων επιτρέπουν τη μάθηση ισχυρών και σύνθετων μοντέλων με 109 έως και 1012 παραμέτρους, στην οποία κλίμακα ένα μόνο απλό μηχάνημα συχνά δεν είναι αρκετό για να παράγει αποτελέσματα σε ένα λογικό χρονικό πλαίσιο. Το πρόβλημα της βελτιστοποίησης αυτών συστημάτων έχει καταστήσει την κατανεμημένη μάθηση ως το βασικό εργαλείο των ερευνητών που επιδιώκουν να λύσουν προβλήματα μηχανικής μάθησης τέτοιου μεγέθους. Ο φόρτος εργασίας χωρίζεται μεταξύ μηχανών – των «workers» (συσκευών που συμμετέχουν στο κατανεμημένο σύστημα και εκτελούν υπολογισμούς), οι οποίες έχουν πρόσβαση σε ένα κοινόχρηστο μοντέλο στο οποίο καταθέτουν τα αποτελέσματα των υπολογισμών που εκτέλεσαν τοπικά με σκοπό να βελτιώσουν την απόδοση του μοντέλου. Ωστόσο η υλοποίηση ενός τέτοιου αποτελεσματικού συστήματος δεν είναι εύκολη. Ένα σημαντικό εμπόδιο είναι η επικοινωνία μεταξύ των συσκευών, εφόσον οι μηχανές πρέπει κάθε φορά που εκτελούν ένα κομμάτι υπολογισμών, να γράφουν στην κοινόχρηστη μνήμη του ενός μηχανήματος σε ρόλο συντονιστή και να αντιγράφουν της μεταβλητές που ανανεώθηκαν από τις υπόλοιπες μηχανές. Αυτό απαιτεί τεράστιο εύρος ζώνης δικτύου για την επικοινωνία μεταξύ των κόμβων, το οποίο μπορεί να δημιουργήσει μεγάλη ποσότητα δικτυακής συμφόρησης σε ένα δίκτυο το οποίο χρησιμοποιείται και για διαφορετικές υπηρεσίες. Επίσης πολλοί αλγόριθμοι εκτελούνται σειριακά και απαιτούν τον συγχρονισμό των μηχανών για να συνεχίζεται η εκτέλεση τους με αποτέλεσμα το σύστημα να είναι τόσο γρήγορο όσο και το πιο αργό μηχάνημα του δικτύου.

Η αρχή έγινε με την βασική ιδέα επίλυσης **blackboard systems** for artificial intelligence, μια προσέγγιση παρόμοιων προβλημάτων κατα την οποία πολλοί πράκτορες (agents) λειτουργούν με μια κοινή βάση πληροφοριών (blackboard), όπου διαβάζουν και γράφουν πληροφορίες σχετικά με την εκτέλεση του αλγορίθμου που τις μοιράζονται με τους υπόλοιπους ομοειδείς τους.

Στην πραγματικότητα οι ρίζες της προσέγγισης αυτής συνδέονται με το πρωτόκολλο **MapReduce** κατα την εκτέλεση του οποίου υπάρχει ένα διμερές γράφημα των αντιστοιχητών (mappers) και των μειωτήρων (reducers) που επεξεργάζονται δεδομένα και το ανασχηματίζουν.

Ένα πρόγραμμα MapReduce αποτελείται από μια μέθοδο-διαδικασία αντιστοίχισης (mapping) η οποία διεξάγει φιλτράρισμα και ταξινόμηση (όπως ταξινόμηση των μαθητών με βάση το όνομά τους σε ουρές, μία ουρά για κάθε όνομα) και μια μέθοδο μείωσης (reduce), η οποία εκτελεί μια συνοπτική λειτουργία υπολογίζοντας τον αριθμό των μαθητών σε κάθε ουρά, δίνοντας συχνότητες εμφάνισης ονομάτων). Το σύστημα MapReduce (που χαρακτηρίζεται ως framework) οργανώνει την επεξεργασία με τη διαλογή των κατανεμημένων διακομιστών (servers), την παράλληλη εκτέλεση των διάφορων εργασιών, τη διαχείριση όλων των επικοινωνιών και μεταφορών δεδομένων μεταξύ των διαφόρων τμημάτων του συστήματος, καθώς και την ανοχή σε σφάλματα (fault tolerance).

Οι βασικές τροποποιήσεις της **αρχιτεκτονικής parameter server** – με την οποία θα ασχοληθούμε στην παρούσα πτυχιακή εργασία – από τα δύο προαναφερθέντα συστήματα είναι ότι ο διακομιστής παραμέτρων είναι σταθερός (persistent). Δηλαδή, σε αντίθεση με το MapReduce, οι διακομιστές διατηρούν την κατάστασή τους, αντί να την καταστρέφουν μετά από κάθε επανάληψη του αλγορίθμου (αυτό καθιστά συστήματα όπως το Hadoop υπερβολικά αργά για τις εφαρμογές μηχανικής μάθησης). Δεύτερον, σε αντίθεση με τα Blackboards, η σύνταξη είναι αρκετά περιορισμένη, αλλά επιτρέπει αποτελεσματική κατανομή της κατάστασης σε πολλές διαφορετικές συνδεδεμένες μηχανές.

Η κατανεμημένη μηχανική μάθηση βασισμένη στον διακομιστή παραμέτρων έχει υιοθετηθεί ευρέως σε πολλές κατανεμημένες πλατφόρμες μάθησης μηχανών. Στην προσπάθεια να δημιουργηθούν πλαίσια και πλατφόρμες για κατανεμημένη μάθηση, έχουν προκύψει ποικίλα έργα αυτής της αρχιτεκτονικής (parameter server), όπως είναι το **ps-lite** από την κοινότητα κατανεμημένης μηχανικής μάθησης (Distributed Machine Learning Community –http://**dmlc**.ml) και το **Multiverso** της Microsoft, που αποτελεί μέρος μια μεγαλύτερης πλατφόρμας deep learning, με όνομα **DMLT** (Microsoft Distributed Machine Learning Toolkit). Αυτό το έργο στοχεύει στη δημιουργία του καλύτερου parameter server framework για την κατανεμημένη μηχανική μάθηση.

Οι απαιτήσεις σχεδιασμού αυτών των λογισμικών στρέφονται σε:

1. Σχεδιασμό ευέλικτων και αποτελεσματικών διεπαφών των parameter servers οι οποίες μπορούν να ενισχύσουν την κατανεμημένη κατάρτιση των υφιστάμενων αλγορίθμων και δικτύων μηχανικής μάθησης με μερικές μόνο γραμμές πρόσθετου κώδικα.

2. Βελτιστοποίηση αλγορίθμων ώστε να παραχθούν καλύτερα αποτελέσματα παράλληλης εκπαίδευσης, π.χ. νέος ασύγχρονος αλγόριθμος, επιταχυνόμενες παράλληλες μέθοδοι βελτιστοποίησης.

3. Να προτείνει καλύτερες μεθόδους παράλληλης κατάρτισης, π.χ. αποτελεσματικότερη κατανομή και προγραμματισμός δεδομένων, αυτόματη ρύθμιση παραμέτρων και μάθηση για τη συγκέντρωση μοντέλου, αποτελεσματική ανταλλαγή δεδομένων για την κατανεμημένη μάθηση.

4. Παροχή καλύτερης υποστήριξης του συστήματος για την προσαρμογή σε διαφορετικά περιβάλλοντα υλικού, π.χ. CPU, GPU, NPU (σε περιπτώσεις φορητών συσκευών και ενσωματωμένων συστημάτων – όπως αναλύεται παρακάτω).

Μια πολύ γνωστή στον χώρο του distributed deep learning βιβλιοθήκη, είναι το **PyTorch**. Πρόκειται για μια βιβλιοθήκη σε γλώσσα python, που είναι κατασκευασμένη με γνώμονα την ταχύτητα εκτέλεσης υπολογισμών σε γράφους, και την επιτάχυνση κατανεμημένης μάθησης με χρήση των πυρήνων των γραφικών (GPUs). Το βασικό πλεονέκτημα του PyTorch είναι οτι υποστηρίζει τη δημιουργία γράφων δυναμικού υπολογισμού (DCG), ενώ οι περισσότερες πλατφόρμες κανανεμημένης μάθησης χρησιμοποιούν στατικούς γράφους (SCG).
Η κύρια χρησιμότητα ενός γράφου δυναμικού υπολογισμού είναι ότι επιτρέπει την επεξεργασία πολύπλοκων εισόδων και εξόδων, χωρίς να δημιουργεί πρόβλημα με την μετατροπή κάθε παρτίδας δεδομένων (batch) εισόδου σε έναν υπέρογκο tensor. Μια μεγάλη χρησιμότητα αυτής της λειτουργίας εμφανίζεται στα επαναλαμβανόμενα νευρωνικά δίκτυα με εισόδους και εξόδους μεταβλητού μήκους. Προκειμένου να επιτευχθεί αυτό στο SCG, θα πρέπει να ορίσουμε τα μεγέθη στις εισόδους και τις εξόδους. Αυτό προκαλεί προβλήματα επειδή, εάν οι αλληλεπιδράσεις είναι πολύπλοκες, τότε η συμπλήρωση ενδέχεται να προσθέσει επιπλέον παραμέτρους στο μοντέλο, οι οποίες ενδέχεται να παρεμβαίνουν. Με την χρήση του DCG αυτό το πρόβλημα προσπερνάται γράφοντας απλά τους υπολογισμούς όπως στην προγραμματιστική λογική ροή. Αυτό γίνεται ακόμα πιο χρήσιμο σε συνελικτικά δίκτυα και αναδρομικά δέντρα (convolutional NNs & recursive tree based NNs.) Το μειονέκτημα της DCG προσέγγισης είναι ότι δεν υποστηρίζει τις διαδικασίες διαχωρισμού των δεδομένων σε παρτίδες (batching) σε περίπλοκες μορφές εισόδου/εξόδου, τόσο αποτελεσματικά όσο το SCG και κατά συνέπεια τείνει να είναι πιο αργή, ειδικά στις GPUs. Επίσης, αν το δίκτυο είναι μεγάλο, τότε η προσέγγιση DCG απαιτεί μεγάλη μνήμη για να αποθηκεύσει τον υπολογιστικό γράφο, το οποίο συνήθως βελτιστοποιείται πολύ σε ένα πλαίσιο SCG.

Η πλέον διαδεδομένη, ταχύτερα αναπτυσσόμενη και υποστηριζόμενη από μεγαλύτερη κοινότητα λύση κατανεμημένης μάθησης, ονομάζεται **Tensorflow**. Πρόκειται για ένα πλαίσιο μάθησης SCG, προϊον της Google, που προσφέρει ποικίλες λύσεις μηχανικής μάθησης και αναλύεται εκτενώς στο κεφάλαιο 2 του παρόντος. Το Tensorflow εκτός από τις βασικές λειτουργικές απαιτήσεις από ένα πλαίσιο μηχανικής μάθησης, όπως είναι η κατανομή σε πολλά μηχανήματα, και η χρήση πολλών διαφορετικών μονάδων υπολογισμών (cpu, gpu, tpu κτλ), προσφέρει πολύ καλά οργανωμένες και συντηρούμενες λύσεις μηχανικής μάθησης για συσκευές μικρότερης επεξεργαστικής ισχύος, όπως κινητές συσκευές (mobile) και ενσωματωμένα συστήματα. Συγκεκριμένα παρέχει έτοιμα πλαίσια, βιβλιοθήκες και διεπαφές, για χρήση προ-εκπαιδευμένων μοντέλων (ανεξαρτήτως του αν εκπαιδεύτηκαν σε κατανεμημένο σύστημα) σε μικρότερες συσκευές. Αυτά αναλύονται στο κεφάλαιο 4 του παρόντος, για συσκευές android & ios – χρήση των Tensorflow Mobile & Tensorflow Lite.

Το **Tensorflow** και το **PyTorch** αποτελούν της δύο πιο δημοφιλείς πλατφόρμες για ανάπτυξη μοντέλων μηχανικής μάθησης. Κάποιες από της διαφορές τους είναι:

- Και τα δύο είναι λογισμικά ανοιχτού κώδικα (open source) και έχουν δημιουργηθεί απο δύο διαφορετικές εταιρίες. Το Tensorflow βασίζεται στο Theano και αναπτύχθηκε απο την ομάδα Google Brain, ενώ το PyTorch που βασίζεται στο Torch έχει αναπτυχθεί από ομάδα της Facebook.
- Η πιο σημαντική τους διαφορά είναι ο τρόπος με τον οποίο δημιουργούν και χειρίζονται τους γραφους τους (computational graphs). Ενώ το Tensorflow δημιουργεί έναν στατικό γράφο, το PyTorch δημιουργεί δυναμικούς. Στο Tensorflow, πρέπει πρώτα να οριστεί ολόκληρος ο γράφος υπολογισμού του μοντέλου και στη συνέχεια να εκτελεστεί το μοντέλο ML, αλλά στο PyTorch ο γράφος μπορεί να ορίζεται και να αλλάζει κατά την διάρκεια της εκτέλεσης. Αυτό είναι εύχρηστο για την εκπαίδευση δικτύων τύπου RNN(Recurrent Neural Networks) που χρείαζεται να επανατροφοδοτούνται με δεδομένα.
- Είναι πιο δύσκολο να εκπαιδευτεί κάποιος ώστε να χρησιμοποιεί το Tensorflow σε σχέση με το PyTorch. Το PyTorch είναι πιο Python-like με αποτέλεσμα τα ML μοντέλα να είναι πιο “intuitive” («ενστικτώδη»). Το Tensorflow απο την αλλή απαιτεί μελέτη πάνω στο χειρισμό του όσον αφορά ορολογίες όπως tf.session και tf.Placeholder, με αποτέλεσμα να καθιστά το Tensorflow πιο δύσκολο στην χρήση του.
- Το Tensorflow έχει πολύ μεγαλύτερη κοινότητα υποστήριξης από το PyTorch, με αποτέλεσμα να είναι πολύ πιο εύκολο να βρει κάποιος πήγες μάθησης και λύσεις πάνω σε πρόβληματα τα οποία προσπαθεί να επιλύσει. Επίσης τα περισσότερα tutorial και online courses καλύπτουν κυρίως το Tensorflow γιατί το PyTorch είναι πιο καινούργιο σαν framework. Άρα από την άποψη βοηθητικού υλικού το Tensorflow είναι σε καλύτερη θέση.
- Το Tensorflow επίσης, διαθέτει το **Tensorboard** σαν επιπλέον εργαλείο του, το οποίο επιτρέπει την απεικόνιση των μοντέλων και των δεδομένων απευθείας σε έναν browser. Το PyTorch δεν προσφέρει εργαλείο απεικόνισης ακόμη αλλα μπορεί κάποιος να απεικονίσει δεδομένα μέσω βιβλιοθηκών σαν το matplotlib της Python. Παρόλα αυτά υπάρχει τρόπος διασύνδεσης του Tensorboard με PyTorch αλλά δεν υποστηρίζεται επίσημα.

Τέλος το Tensorflow θεωρείται πολύ καλύτερο για μοντέλα τα οποία θα χρησιμοποιηθούν στην βιομηχανία και θα είναι μεγαλύτερου μεγέθους. Από την άλλη πλευρά το PyTorch είναι πιο εύκολο και ελαφρύ για για χρήση από κάποιον νέο στην μηχανική μάθηση που θέλει να κάνει κάτι γρήγορο, απλό και μικρό σε μέγεθος.

1.2 ΑΛΓΟΡΙΘΜΟΙ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Οι αλγόριθμοι της Μηχανικής Μάθησης χωρίζονται σε δυο μεγάλες κατηγορίες, τους αλγόριθμους **με επίβλεψη** κι αυτούς **χωρίς επίβλεψη**.

- **Αλγόριθμοι με επίβλεψη**

Ίσως ο πιο διαδεδομένος τύπος μάθησης με πληθώρα εφαρμογών στο πλαίσιο της αναγνώρισης προτύπων. Οι αλγόριθμοι με επίβλεψη απαιτούν την παρουσία του ανθρώπινου παράγοντα που

θα προσδιορίσει τόσο τα δεδομένα εισόδου όσο και τα δεδομένα εξόδου, καθώς πρέπει να παρέχει και ανάδραση σχετικά με την ακρίβεια των προβλέψεων (accuracy of predictions) κατα την διάρκεια της εκπαίδευσης. Με την ολοκλήρωση της εκπαίδευσης, ο αλγόριθμος εφαρμόζει οτι έμαθε σε νέα δεδομένα.

- **Αλγόριθμοι χωρίς επίβλεψη**

Οι αλγόριθμοι χωρίς επίβλεψη απαιτούν τα δεδομένα εισόδου, όχι όμως τις τιμές εξόδου, είτε επειδή είναι άγνωστες, είτε επειδή είναι αρκετά δύσκολο να υπολογιστούν. Αντ' αυτού, χρησιμοποιούν μια επαναληπτική προσέγγιση που ονομάζεται «βαθιά μάθηση» (deep learning)* για να εξάγουν αποτελέσματα μέσω ανασκόπησης των δεδομένων. Οι αλγόριθμοι χωρίς επίβλεψη χρησιμοποιούνται για πιο πολύπλοκες εργασίες από οτι οι αλγόριθμοι με επίβλεψη. Υπάρχουν πολλές και ποικίλες περιπτώσεις που χαρακτηρίζονται ως «χωρίς επίβλεψη». Αρκετά συνηθισμένες περιπτώσεις είναι η κατανομή πιθανότητας, η ομαδοποίηση και η συμπίεση δεδομένων.

- **Αλγόριθμοι με μερική επίβλεψη**

Στην περίπτωση της μερικής επίβλεψης είναι γνωσοί μερικοί στόχοι, αλλά όχι όλοι. Αυτό το ενδεχόμενο βρίσκεται μεταξύ μάθησης με επίβλεψη και μάθησης χωρίς επίβλεψη, αφού συχνά χρησιμοποιούνται ένα σύνολο δεδομένων για εκπαίδευση (train set) με γνωστούς στόχους (labels) και ένα σύνολο δεδομένων ελέγχου (test set) με άγνωστες εξόδους.Ο σκοπός σε αυτή την περίπτωση είναι να προβλεφθούν οι στόχοι του συνόλου δεδομένων ελέγχου. Άλλη, σπανιότερη περίπτωση μάθησης με μερική επίβλεψη είναι αυτή στην οποία εισάγονται περιορισμοί, όπως για παράδειγμα η ανάγκη μερικά πρότυπα να έχουν ίδιους στόχους μεταξύ τους, χωρίς να είναι εκ των προτέρων γνωστοί οι στόχοι αυτοί.

Οι διαδικασίες που εμπλέκονται στην Μηχανική Μάθηση είναι παρόμοιες με αυτές της εξόρυξης δεδομένων και της πρότυπης μοντελοποίησης. Σε όλες τις παραπάνω απαιτείται έρευνα σε όγκους δεδομένων με σκοπό την εύρεση προτύπων, και επαναπροσδιορισμού των ενεργειών του προγράμματος αναλόγως.

Δεδομένα και χρήση στην καθημερινότητα

Πολλοί άνθρωποι χρησιμοποιούν αλγορίθμους Μηχανικής Μάθησης καθημερινά, όταν πραγματοποιούν αγορές μέσω internet και παρακολουθούν διαφημίσεις σχετικές με το αντικείμενο που θέλουν να αγοράσουν. Αυτό συμβαίνει επειδή οι «μηχανές συστάσεων» χρησιμοποιούν την Μηχανική Μάθηση για να εξατομικεύσουν τις διαφημίσεις και να βελτιώσουν την εμπειρία χρήσης σε πραγματικό χρόνο.

Εκτός από την εξατομίκευση διαφημίσεων και άλλες χρήσεις στο εμπόριο, στην καθημερινότητα συναντούμε κι άλλες χρήσεις αλγορίθμων Μηχανικής Μάθησης, όπως αναγνώριση απάτης (fraud detection), φιλτράρισμα ανεπιθύμητων μηνυμάτων ηλεκτρονικού ταχυδρομείου (spam), ασφάλεια δικτύων, πρόβλεψη συντήρησης και ροές ειδήσεων. Για παράδειγμα, η ροή ειδήσεων

(news feed) στα μέσα κοινωνικής δικτύωσης, δημιουργείται ανάλογα με την δραστηριότητα του κάθε χρήστη. Όταν ο χρήστης αφιερώνει χρόνο για να διαβάσει κάποιο περιεχόμενο, ή αλληλεπιδρά με αυτό, η ροή ειδήσεων εμπλουτίζεται με περισσότερο υλικό σχετικό με το ενδιαφέρον του χρήστη εκείνη την ώρα. Στο παρασκήνιο χρησιμοποιούνται αλγόριθμοι στατιστικής ανάλυσης και πρόβλεψης για να αναγνωρίσουν πρότυπα στα δεδομένα και την συμπεριφορά του χρήστη και να δημιουργήσουν ροές με ανάλογα εξατομικευμένο περιεχόμενο.

1.3 ΒΑΣΙΚΕΣ ΕΦΑΡΜΟΓΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Όλοι οι αλγόριθμοι μηχανικής μάθησης δέχονται κάποιον τύπο δεδομένων σαν είσοδο, αλλά επιτυγχάνουν διαφορετικό σκοπό. Ανάλογα με αυτό τον σκοπό, κατατάσσονται στις εξής κατηγορίες:

Ταξινόμηση (Classification)

Ο σκοπός των αλγορίθμων ταξινόμησης είναι να αναγνωρίσουν και να ταξινομήσουν δείγματα, ανάλογα με την τάξη στην οποία ανήκουν. Ένα γνωστό παράδειγμα, είναι εικόνες διαφορετικών ζώων, όπου το σύστημα καλείται να τα ταξινομίσει ανάλογα με το είδος τους, λόγου χάρη σε γάτες, σκύλους, πρόβατα κτλ. Στην πράξη, αυτοί οι αλγόριθμοι χρησιμοποιούνται για να ταξινομήσουν φωνή (voice classification), ή να αναγνωρίσουν αντικείμενα (object recognition). Στις περισσότερες περιπτώσεις το πλήθος των κλάσεων είναι εκ των προτέρων γνωστό και μπορεί να είναι από δύο μέχρι μερικές χιλιάδες. Οι περιπτώσεις στις οποίες οι κλάσεις (ή κατηγορίες) είναι δύο, ονομάζονται και δυο-κλάσεων (two-class) ή δυαδικά (binary) προβλήματα ταξινόμησης. Πρόκειται για μια διαδικασία μάθησης **με επίβλεψη**, αφού ο αλγόριθμος απαιτείται να εκπαιδευτεί πρώτα με σύνολα δεδομένων εισόδου, και τις αντίστοιχες κλάσεις στις οποίες ανήκουν (labels).

Εικόνα 2: Ταξινόμηση – Ονόματα κλάσεων (labels) και ακρίβεια (accuracy)

Υπάρχουν πολλοί τρόποι για να ελεγθεί η ικανότητα σωστής πρόβλεψης ενός μοντέλου ταξινόμησης, όμως ο πιο κοινός είναι ο υπολογισμός της **ακρίβειας (accuracy)** της ταξινόμησης. Η ακρίβεια είναι το ποσοστό των σωστών προβλέψεων προς τις συνολικές

προβλέψεις του μοντέλου. Για παράδειγμα, εάν ένα μοντέλο έκανε 5 προβλέψεις συνολικά και οι 3 από αυτές ήταν σωστές ενώ οι 2 λανθασμένες, η ακρίβεια θα προκύψει από τα παρακάτω:

```
accuracy = correct predictions / total predictions * 100
accuracy = 3 / 5 * 100
accuracy = 60%
```

Εικόνα 3: Υπολογισμός της ακρίβειας (accuracy)

Παλινδρόμηση (Regression)

Τα μοντέλα παλινδρόμησης (linear regression, regression trees, support vector regression, κ.α.) έχουν σκοπό να υπολογίσουν μια συνάρτηση (f) που θα αντιστοιχίζει συνεχείς τιμές εξόδου (y) βάσει ενός συνόλου παραμέτρων που δέχονται ως είσοδο (x). Για παράδειγμα, ο αλγόριθμος μπορεί δοθέντων συνόλων δεδομένων για την ατμοσφαιρική πίεση, την υγρασία και το υψόμετρο μια περιοχής, να εξαγάγει τιμές για την θερμοκρασία στην περιοχή αυτή. Αποτελεί επίσης διαδικασία μάθησης **με επίβλεψη**, αφού για να εκπαιδευτεί το μοντέλο χρειάζεται σύνολο δεδομένων εισόδου **και** εξόδου (πχ τιμές για πίεση, υγρασία, υψόμετρο **και** θερμοκρασία για το σύνολο δεδομένων εκπαίδευσης).

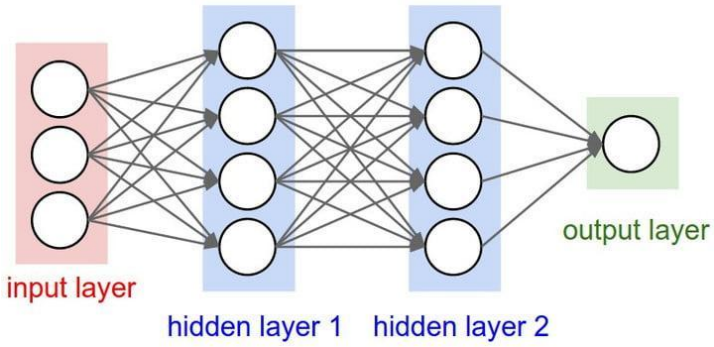
Ομαδοποίηση (Clustering)

Τα μοντέλα ομαδοποίησης αναθέτουν ένα σύνολο χαρακτηριστικών/ιδιοτήτων σε υποσύνολα του συνόλου δεδομενων εισόδου έτσι ώστε να μπορούν να διαχωριστούν σε ομάδες (clusters) βάσει αυτών των χαρακτηριστικών. Οι αλγόριθμοι ομαδοποίησης δεν εξάγουν συνεχεία / διακριτές τιμές εντος ενός συνόλου και δεν δέχονται σύνολα τιμών εξόδου κατα την εκπαίδευση του μοντέλου. Συνεπώς πρόκειται για διαδικασία μάθησης **χωρίς επίβλεψη (unsupervised)**.

1.3.1 ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Τα Τεχνητά Νευρωνικά Δίκτυα είναι ένα από τα βασικά εργαλεία που χρησιμοποιούνται στην Μηχανική Μάθηση. Όπως περιγράφει ο όρος «νευρωνικά» είναι εμπνευσμένα από τον ανθρώπινο εγκέφαλο, και δημιουργήθηκαν με σκοπό να αντιγράψουν τον τρόπο που μαθαίνει ένας άνθρωπος. Τα νευρωνικά δίκτυα αποτελούνται από επίπεδα (layers) εισόδου και εξόδου, καθώς και ένα ή περισσότερα «κρυφά επίπεδα» (hidden layers) τα οποία είναι υπεύθυνα για τον μετασχηματισμό των δεδομένων εισόδου σε κατάλληλη μορφή για να χρησιμοποιηθούν από το επόμενο επίπεδο. Χρησιμοποιούνται κυρίως για αναγνώριση προτύπων που είναι πολλά σε αριθμό ή υπερβολικά πολύπλοκα για να αναγνωριστούν/επιλυθούν από τον άνθρωπο. Παρ'ότι τα νευρωνικά δίκτυα έχουν εμφανιστεί από την δεκαετία του 1940, μόλις τις τελευταίες δεκαετίες έχουν χρησιμοποιηθεί στην τεχνητή νοημοσύνη, κι αυτό εξαιτίας της τεχνικής που ονομάζεται «οπισθοδιάδοση» (back propagation), η οποία επιτρέπει στα δίκτυα να αναπροσδιορίζουν κατάλληλα την δομή των κρυφών επιπέδων τους ή τους νευρώνες, όταν τα δεδομένα εξόδου δεν είναι τα επιθυμητά, για παράδειγμα σε ένα δίκτυο που αναγνωρίζει σαν πρότυπο μια γάτα, ενώ θα έπρεπε να αναγνωρίσει έναν σκύλο.

Εάν για παράδειγμα, ένα δίκτυο έχει κατασκευαστεί για να αναγνωρίζει ένα αντικείμενο μέσα σε μια εικόνα (object detection), πιθανόν το πρώτο δίκτυο να αναλύει την φωτεινότητα των εικονοστοιχείων (pixels).



Εικόνα 4: Παράδειγμα ΤΝΔ με (2) κρυφά επίπεδα

Τα επόμενα κρυφά επίπεδα, για την ακρίβεια μια σειρά από επόμενα επίπεδα, θα μπορούσαν να αναγνωρίσουν τις γωνίες που υπάρχουν στην εικόνα, βασισμένα στις γραμμές που βρίσκουν σε παρόμοια εικονοστοιχεία. Αργότερα, άλλα κρυφά επίπεδα αναγνωρίζουν υφές και σχήματα κια ούτω καθεξής. Όταν φτάσουν τα δεδομένα στο τέταρτο – πέμπτο επίπεδο (ή ομάδες κρυφών επιπέδων) το δίκτυο πλέον θα μπορεί να αναγνωρίσει πολλά σύνθετα χαρακτηριστικά στην εικόνα, όπως λ.χ. μάτια, μύτη, στόμα, τα οποία συνήθως συνυπάρχουν σε μικρή απόσταση το ένα από το άλλο.
Όταν ολοκληρωθεί αυτή η διαδικασία, ο εκπαιδευτής του δικτύου μπορεί να προσθέσει «ετικέτες – labels» στα δεδομένα εξόδου και με την χρήση της τεχνικής back propagation να μάθει στον αλγόριθμο να αυτο-διορθώνεται. Μετά την διαδικασία της εκπαίδευσης, το δίκτυο μπορεί να λειτουργεί και να ταξινομεί μόνο του δεδομένα, χωρίς ανθρώπινη υποστήριξη.

1.3.2 ΕΙΔΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

Η πιο βασική μορφή ΤΝΔ είναι αυτή του δικτύου πρόσθιας τροφοδότησης (**feedforward** neural network) στην οποία η ροή των δεδομένων έχει μόνο μια κατεύθυνση, από το επίπεδο εισόδου προς το επίπεδο εξόδου. Μια άλλη μορφή, η οποία χρησιμοποιείται κυρίως σε δυσκολότερες και πιο πολύπλοκες διαδικασίες μάθησης, όπως αναγνώριση χειρογράφων ή γλώσσας, είναι τα δίκτυα ανατροφοδότησης (feedback ή **recurrent**). Σε αυτά η ροή των δεδομένων είναι αμφίδρομη. Η πιο διαδεδομένη μορφή στις κοινές εφαρμογές της ΜΜ στην καθημερινότητά μας, είναι τα συνελικκτικά νευρωνικά δίκτυα **(Convolutional Neural Networks – CNN)** και αυτό επειδή λειτουργούν με εικόνες ως δεδομένα εισόδου, γεγονός που επιτρέπει την δημιουργία κατάλληλης δομής των κρυφών επιπέδων εκ των προτέρων. Αυτό βελτιώνει κατα πολύ την πρόσθια ροή (λιγότερες διορθώσεις) και μειώνουν σημαντικά τις παραμέτρους στο δίκτυο.

Η επιλογή του καταλληλότερου τύπου δικτύου για μια εφαρμογή, εξαρτάται από την φύση της ίδιας της εφαρμογής και των δεδομένων εισόδου. Πολλές φορές μπορεί να χρησιμοποιηθεί συνδυασμός διαφορετικών τύπων, για παράδειγμα σε ιδιαίτερες και πολύπλοκες διαδικασίες όπως είναι η αναγνώριση φωνής.

1.3.3 ΣΥΝΕΛΙΚΤΙΚΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Όπως γνωρίσαμε νωρίτερα, τα ΝΔ δέχονται ως είσοδο ένα διάνυσμα και το μετασχηματίζουν κατάλληλα με την χρήση μια σειράς *κρυφών δικτύων*. Κάθε ένα δίκτυο από αυτά αποτελείται από ένα σύνολο νευρώνων, όπου ο κάθε ένας συνδέεται πλήρως με όλους τους νευρώνες του προηγούμενου επιπέδου, και λειτουργεί εντελώς ανεξάρτητα από τους υπόλοιπους νευρώνες του ίδιου επιπέδου. Το τελευταίο επίπεδο – που είναι πλήρως συνδεδεμένο με τους νευρώνες του προηγούμενου επιπέδου, καλείται «επίπεδο εξόδου» και αναπαριστά τις κλάσεις εξόδου.

Τα συνελικτικά δίκτυα χρησιμοποιούνται κατά κύριο λόγο για την ταξινόμηση εικόνων, συσσώρευση σε ομάδες με όμοια χαρακτηριστικά (αναζήτηση φωτογραφιών) και εκτελούν αναγνώριση αντικειμένων μέσα σε σκηνές (scenes). Είναι αλγόριθμοι που μπορούν να εντοπίσουν πρόσωπα, άτομα, οδική σήμανση, όγκους και πολλές άλλες πτυχές οπτικών δεδομένων.

Τα δίκτυα αυτά εκτελούν οπτική αναγνώριση χαρακτήρων (OCR) για την ψηφιοποίηση κειμένου και την επεξεργασία φυσικής γλώσσας σε αναλογικά και χειρόγραφα έγγραφα, όπου οι εικόνες είναι σύμβολα που πρέπει να μεταγραφούν. Τα CNN μπορούν επίσης να εφαρμοστούν σε δεδομένα ήχου όταν αυτά παρουσιάζονται οπτικά σαν φασματογράφημα. Η αποτελεσματικότητα των συνελικτικών δικτύων στην αναγνώριση εικόνας είναι ένας από τους κύριους λόγους για τους οποίους ο κόσμος έχει αντιληφθεί την αποτελεσματικότητα της βαθιάς μάθησης γενικότερα. Σηματοδοτούν σημαντικές εξελίξεις στην «υπολογιστική όραση» (computer vision – cv), η οποία έχει εμφανείς εφαρμογές σε αυτο-οδηγούμενα αυτοκίνητα, ρομποτική, drones, ασφάλεια, ιατρικές διαγνώσεις και θεραπείες για άτομα με προβλήματα όρασης.

Τα συνελικτικά αναδιαμορφώνουν και επεξεργάζονται τις εικόνες ως tensors, δηλαδή πίνακες αριθμών με πρόσθετες διαστάσεις. Στην πράξη είναι δύσκολο να απεικονιστούν, οπότε συνηθίζουμε να τα προσεγγίζουμε κατ 'αναλογία. Ένα **scalar** είναι απλά ένας αριθμός, όπως το 7. Ένα **vector** (διάνυσμα) είναι μια λίστα-πίνακας αριθμών (π.χ. [7,8,9]), και ένα **matrix** (μήτρα) είναι ένα ορθογώνιο πλέγμα αριθμών που καταλαμβάνουν πολλές σειρές και στήλες όπως ένα υπολογιστικό φύλλο. Γεωμετρικά, εάν το scalar είναι ένα σημείο μηδενικής διάστασης, τότε ένα διάνυσμα είναι μια μονοδιάστατη γραμμή, μια μήτρα είναι ένα δισδιάστατο επίπεδο, μια στοίβα μήτρων είναι ένας τρισδιάστατος κύβος και όταν κάθε στοιχείο αυτών των μητρών έχει μια στοίβα αντιστοιχούμενων χαρακτηριστικών που είναι συνδεδεμένα με αυτήν, εισάγεται η τέταρτη διάσταση. Για παράδειγμα, αυτός είναι ένας πίνακας 2 x 2:

$$\begin{bmatrix} 1, 2 \\ 5, 8 \end{bmatrix}$$

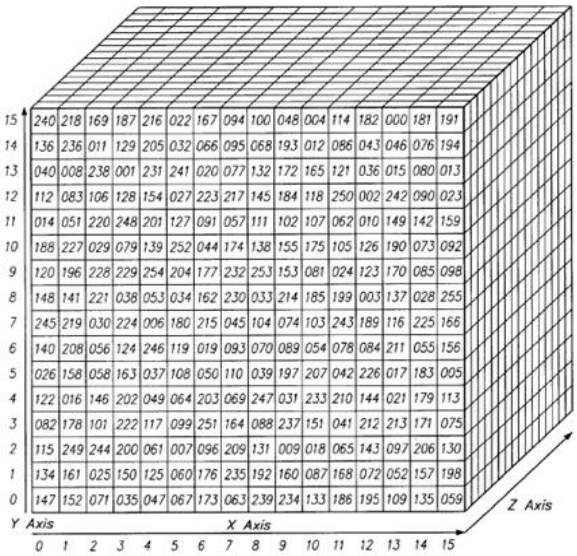
Ένα tensor περιλαμβάνει τις διαστάσεις πέραν αυτού του διδιάστατου επιπέδου. Ένα τρισδιάστατο tensor μπορεί εύκολα να απεικονιστεί, με μια σειρά αριθμών που είναι τοποθετημένοι σε έναν κύβο. Εδώ παρουσιάζεται ένα tensor 2 x 3 x 2 (απεικονίζουμε το κάτω στοιχείο από κάθε πίνακα 2-στοιχείων να εκτείνετε κατα μήκος του άξονα z, για να εξηγήσουμε εμπειρικά τι σημαίνει τρισδιάστατος πίνακας):

$$\begin{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 5 \end{pmatrix} & \begin{pmatrix} 4 \\ 7 \end{pmatrix} \\ \begin{pmatrix} 3 \\ 4 \end{pmatrix} & \begin{pmatrix} 4 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 8 \end{pmatrix} \end{pmatrix}$$

Το ίδιο παράδειγμα σε κώδικα γράφεται με τον εξής τρόπο:

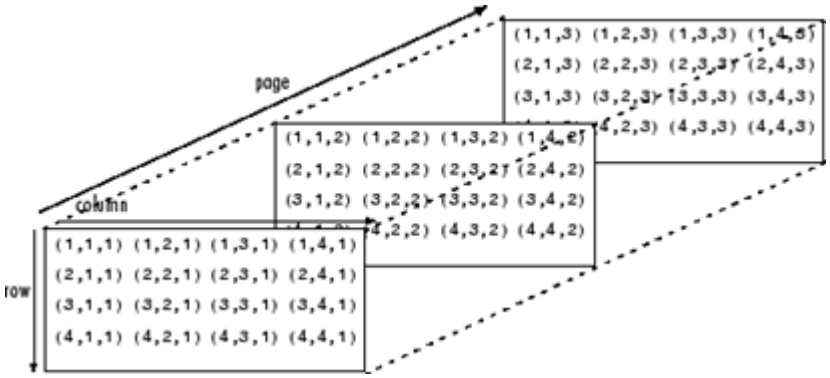
```
[[[2,3],[3,5],[4,7]],[[3,4],[4,6],[5,8]]]
```

Ενώ οπτικά θα φαίνεται σαν την παρακάτω εικόνα:



Εικόνα 5: Οπτικοποίηση 3-διάστατου πίνακα

Με άλλα λόγια, οι tensors σχηματίζονται από συστοιχίες πινάκων που είναι εμφωλευμένες μέσα σε πίνακες και αυτή η «εμφώλευση» μπορεί να φτάσει σε πολλά επίπεδα βάθος, αντιπροσωπεύοντας έναν αυθαίρετο αριθμό διαστάσεων πολύ μεγαλύτερο από αυτό που μπορούμε να απεικονίσουμε χωρικά. Ένας 4-D tensor απλώς θα αντικαταστήσει κάθε ένα από αυτά τα scalars με έναν πίνακα που έχει τοποθετηθεί ένα επίπεδο βαθύτερα. Τα συνελικτικά δίκτυα ασχολούνται με 4-D tensors όπως αυτό που ακολουθεί (παρατηρήστε το εμφωλευμένο πίνακα (nested array)).



Εικόνα 6: Οπτικοποίηση ενός 4D tensor

Το πλάτος και το ύψος μιας εικόνας είναι έννοιες εύκολα κατανοητές, όμως το βάθος είναι επίσης απαραίτητο λόγω του τρόπου με τον οποίο κωδικοποιούνται τα χρώματα στην ψηφιακή μορφή. Η κωδικοποίηση RGB, για παράδειγμα, παράγει μια εικόνα με βάθος τριών στρώσεων. Κάθε στρώμα ονομάζεται "κανάλι" και μέσω της συνέλιξης παράγει μια στοίβα χαρτών χαρακτηριστικών (που εξηγούνται παρακάτω), τα οποία υπάρχουν στην τέταρτη διάσταση. Τα χαρακτηριστικά είναι απλώς λεπτομέρειες των εικόνων, όπως μια γραμμή ή καμπύλη, για τα οποία τα συνελικτικά δίκτυα δημιουργούν χάρτες.

Πως λειτουργούν τα συνελικτικά δίκτυα

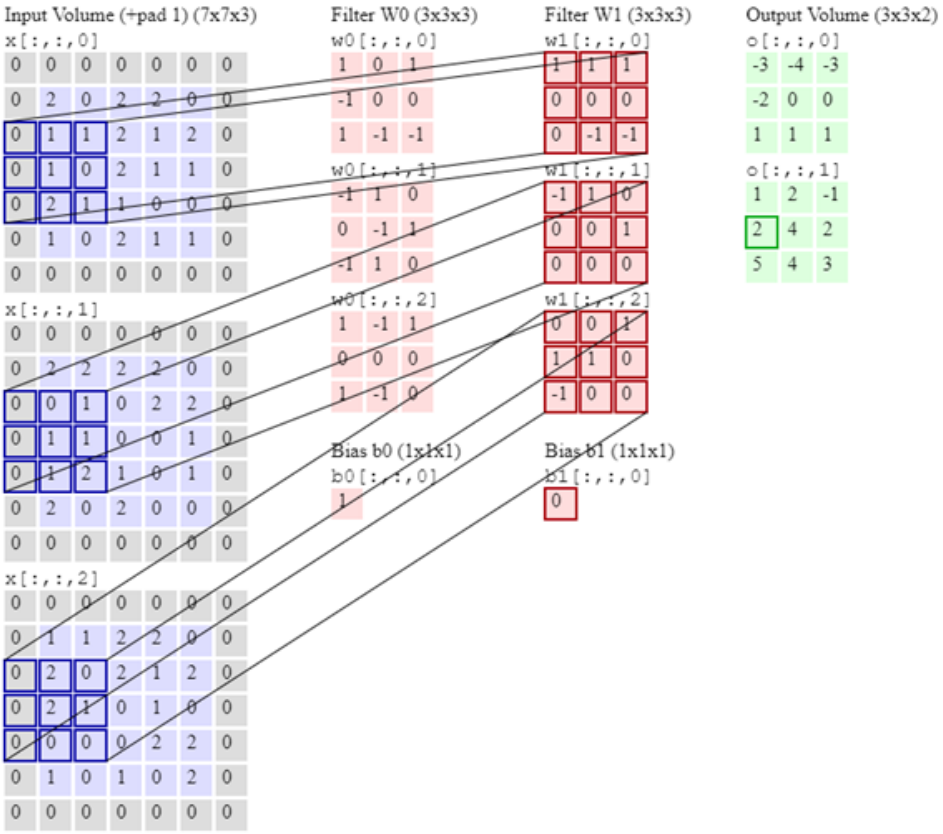
Το πρώτο πράγμα που πρέπει να γνωρίζει κανείς για τα συνελικτικά δίκτυα είναι ότι δεν αντιλαμβάνονται τις εικόνες όπως οι άνθρωποι. Επομένως, θα πρέπει να σκεφτούμε με διαφορετικό τρόπο τι σημαίνει μια εικόνα, καθώς αυτή τροφοδοτείται και επεξεργάζεται από ένα συνελικτικό δίκτυο. Τα συνελικτικά δίκτυα αντιλαμβάνονται τις εικόνες ως όγκους, δηλαδή τρισδιάστατα αντικείμενα, αντί να επιμετρώνται σε επίπεδο καμβά που έχει μόνο πλάτος και ύψος. Αυτό οφείλεται στο γεγονός ότι οι ψηφιακές έγχρωμες εικόνες έχουν κωδικοποίηση RGB, ανάμειξη αυτών των τριών χρωμάτων για να παράγουν το φάσμα χρωμάτων που οι άνθρωποι αντιλαμβάνονται. Ένα συνελικτικό δίκτυο καταλαμβάνει τέτοιες εικόνες ως τρία ξεχωριστά στρώματα χρώματος στοιβαγμένα το ένα πάνω στο άλλο.

Επομένως, ένα συνελικτικό δίκτυο λαμβάνει μια κανονική έγχρωμη εικόνα ως ορθογώνιο κιβώτιο του οποίου το πλάτος και το ύψος μετρούνται με τον αριθμό των εικονοστοιχείων κατά μήκος αυτών των διαστάσεων και του οποίου το βάθος είναι τρία επίπεδα, ένα για κάθε γράμμα από τα RGB (δηλαδή για κάθε χρώμα). Αυτά τα επίπεδα βάθους όπως αναφέραμε νωρίτερα ονομάζονται κανάλια (“channels”). Καθώς οι εικόνες επεξεργάζονται μέσω ενός συνελικτικού δικτύου, θα τις περιγράψουμε με όρους όγκων εισόδου και εξόδου, εκφράζοντάς τους μαθηματικά, ως μήτρες πολλαπλών διαστάσεων με την μορφή: 30x30x3.

Από το ένα επίπεδο στο άλλο, οι διαστάσεις τους αλλάζουν για λόγους που θα εξηγηθούν παρακάτω. Χρειάζεται ιδιαίτερη προσοχή στα ακριβή μέτρα κάθε διάστασης του όγκου της εικόνας, επειδή αποτελούν τη βάση των λειτουργιών γραμμικής άλγεβρας που χρησιμοποιούνται για την επεξεργασία των εικόνων. Για κάθε εικονοστοιχείο μιας εικόνας, η ένταση των R, G και B θα εκφράζεται με έναν αριθμό, και αυτός ο αριθμός θα είναι ένα στοιχείο σε μία από τις τρεις, στοιβαγμένες διδιάστατες μήτρες, οι οποίες μαζί σχηματίζουν τον όγκο της εικόνας.

Αυτοί οι αριθμοί είναι τα αρχικά, ακατέργαστα, αισθητήρια χαρακτηριστικά που τροφοδοτούνται στο συνελικτικό δίκτυο ως «είσοδος» και ο σκοπός του εκάστοτε συνελικτικού δικτύου είναι να βρεί ποιος από αυτούς τους αριθμούς είναι σημαντικά μηνύματα που πραγματικά βοηθούν στην ταξινόμηση των εικόνων με μεγαλύτερη ακρίβεια.

Αντί να επικεντρωθεί σε ένα εικονοστοιχείο τη φορά, ένα συνελικτικό δίκτυο παίρνει τετράγωνα κομμάτια των pixel (patches) και τα περνάει μέσα από ένα φίλτρο. Αυτό το φίλτρο είναι επίσης τετράγωνο πλέγμα μικρότερο από την ίδια την εικόνα και ίσο σε μέγεθος με το κάθε κομμάτι.



Εικόνα 7: Παράδειγμα φίλτρων στα pixels της εικόνας

Έχουμε για παράδειγμα 2 μήτρες. Η μία έχει διαστάσεις 30x30 και η άλλη 3x3. Δηλαδή, το φίλτρο καλύπτει το ένα εκατοστό του εμβαδού επιφάνειας **ενός καναλιού** της εικόνας.

Κρατάμε το προϊόν του φίλτρου με το εκάστοτε κομμάτι του καναλιού της εικόνας. Εάν οι δύο μήτρες έχουν υψηλές τιμές στις ίδιες θέσεις, η έξοδος του τελικού προϊόντος θα είναι υψηλή. Εάν δεν το κάνουν, θα είναι χαμηλή. Με αυτόν τον τρόπο, μια μοναδική τιμή - η έξοδος του

κουκκιδωτού προϊόντος (dotted product) - μπορεί να μας πει αν το σχέδιο εικονοστοιχείων στην υποκείμενη εικόνα ταιριάζει με το πρότυπο εικονοστοιχείων που εκφράζεται από το φίλτρο μας.

Για να εξυπηρετήσουμε το παράδειγμα, υποθέτουμε ότι το φίλτρο μας εκφράζει μια οριζόντια γραμμή, με υψηλές τιμές στη δεύτερη σειρά και χαμηλές τιμές στην πρώτη και την τρίτη σειρά. Αν θεωρήσουμε ότι αρχίζουμε στην επάνω αριστερή γωνία της υποκείμενης εικόνας και μεταφέρουμε το φίλτρο κατά μήκος της εικόνας βήμα προς βήμα μέχρι να φτάσει στην άνω δεξιά γωνία. Η ποσότητα των εικονοστοιχείων που θα μετακινηθούν είναι γνωστή ως **βήμα (stride)**. Μπορεί να μετακινήσουμε το φίλτρο στην αμέσως επόμενη στήλη εαν το βήμα είναι ίσο με 1, ή να προχωρήσουμε με μεγαλύτερα «βήματα».

Σε κάθε βήμα, παίρνουμε ένα άλλο αποτέλεσμα και τοποθετούμε τα αποτελέσματα αυτού του προϊόντος σε μια τρίτη μήτρα γνωστή ως χάρτης ενεργοποίησης (activation map). Το πλάτος ή ο αριθμός των στηλών του χάρτη ενεργοποίησης ισούται με τον αριθμό των βημάτων που παίρνει το φίλτρο για να διασχίσει την υποκείμενη εικόνα. Δεδομένου ότι τα μεγαλύτερα βήματα οδηγούν σε λιγότερες επαναλήψεις, ένα μεγάλο βήμα θα παράγει ένα μικρότερο χάρτη ενεργοποίησης. Αυτό είναι σημαντικό, διότι το μέγεθος των πινάκων που επεξεργάζονται και παράγουν τα συνελικτικά δίκτυα σε κάθε στρώμα είναι άμεσα αναλογικά με το πόσο υπολογιστικά δαπανηρά είναι και πόσο χρόνο χρειάζονται για την εκπαίδευση. Ένα μεγαλύτερο βήμα σημαίνει λιγότερο χρόνο για υπολογισμούς στην διαδικασία της εκπαίδευσης.

Ένα φίλτρο τοποθετημένο επάνω στις τρεις πρώτες σειρές ολισθαίνει επάνω τους και στη συνέχεια θα ξεκινήσει πάλι με τις γραμμές 4-6 της ίδιας εικόνας. Εάν έχει ένα βήμα ίσο με 3, τότε θα παράγει μια μήτρα των «κουκκιδωτών» προϊόντων με μέγεθος 10x10. Το ίδιο φίλτρο που αντιπροσωπεύει μια οριζόντια γραμμή μπορεί να εφαρμοστεί και στα τρία κανάλια της υποκείμενης εικόνας, R, G και B. Και οι τρεις χάρτες ενεργοποίησης 10x10 μπορούν να προστεθούν μαζί, έτσι ώστε ο συνολικός χάρτης ενεργοποίησης μιας οριζόντιας γραμμής και στα τρία κανάλια της υποκείμενης εικόνας είναι επίσης 10x10 (ο μέσος όρος τους).

Στην πράξη, επειδή οι εικόνες έχουν πολλές γραμμές που πηγαίνουν προς πολλές διαφορετικές κατευθύνσεις και περιέχουν πολλά διαφορετικά είδη σχημάτων και μοτίβων εικονοστοιχείων, θα χρειαστεί να περάσουν άλλα φίλτρα σε όλη την υποκείμενη εικόνα για να αναζητήσουμε αυτά τα μοτίβα. Θα μπορούσαμε, για παράδειγμα, να αναζητήσουμε 96 διαφορετικά μοτίβα στα εικονοστοιχεία. Αυτά τα 96 πρότυπα θα δημιουργήσουν μια στοίβα από 96 χάρτες ενεργοποίησης, με αποτέλεσμα ένα νέο όγκο 10x10x96.

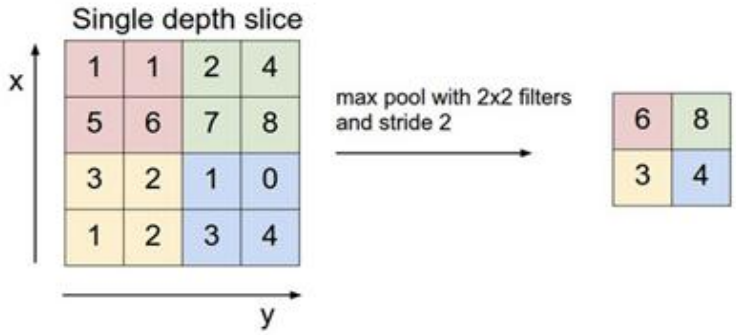
Ένα από τα κύρια προβλήματα με τις εικόνες είναι ότι συνήθως έχουν μεγάλες διαστάσεις, πράγμα που σημαίνει ότι κοστίζουν πολύ χρόνο και υπολογιστική ισχύ για επεξεργασία. Τα συνελικτικά δίκτυα έχουν σχεδιαστεί για να μειώνουν τις διαστάσειςτων εικόνων με διάφορους τρόπους. Το βήμα του φιλτραρίσματος είναι ένας τρόπος. Ένας άλλος τρόπος είναι μέσω της **δειγματοληψίας**.

Μέγιστη συγκέντρωση / Δειγματοληψία

(Max pooling / Downsampling)

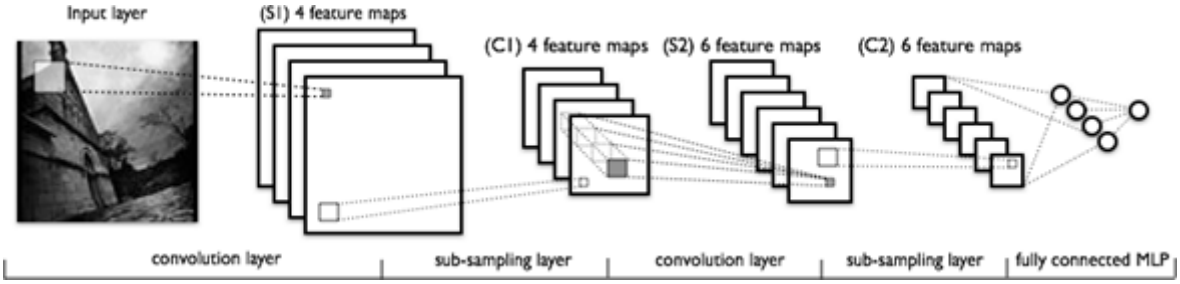
Το ακόλουθο στρώμα σε ένα συνελικτικό δίκτυο έχει δυο ονόματα: μέγιστη συγκέντρωση (max pooling) και υποδειγματοληψία (downsampling). Οι χάρτες ενεργοποίησης τροφοδοτούνται σε ένα στρώμα δειγματοληψίας και, όπως και οι συνελίξεις, αυτή η μέθοδος εφαρμόζει ένα patch τη

φορά. Σε αυτήν την περίπτωση, η μέγιστη συγκέντρωση λαμβάνει απλώς τη μεγαλύτερη τιμή από ένα patch μιας εικόνας, την τοποθετεί σε μια νέα μήτρα δίπλα στις μέγιστες τιμές από άλλα patches και απορρίπτει τις υπόλοιπες πληροφορίες που περιέχονται στους χάρτες ενεργοποίησης.



Εικόνα 8: Μέγιστη συγκέντρωση - Max pooling

Μόνο οι θέσεις στην εικόνα που έδειξαν την ισχυρότερη συσχέτιση με κάθε χαρακτηριστικό (μέγιστη τιμή) διατηρούνται και αυτές οι μέγιστες τιμές συνδυάζονται για να σχηματίσουν έναν μικρότερο χώρο. Πολλές πληροφορίες σχετικά με τις μικρότερες τιμές χάνονται σε αυτό το βήμα, γεγονός που έχει ωθήσει την έρευνα σε εναλλακτικές μεθόδους. Παρ'αυτα η δειγματοληψία έχει το πλεονέκτημα, ακριβώς λόγω της απώλειας πληροφοριών, της μείωσης της απαιτούμενης αποθήκευσης και επεξεργασίας.



Εικόνα 9: Ακολουθία μετασχηματισμών που εμπλέκονται σε ένα τυπικό συνελικτικό δίκτυο.

1.4 ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ ΣΤΟ ΝΕΦΟΣ (MLaaS)

Η ανάπτυξη ενός προϊόντος σε ολοκληρωμένη υπηρεσία στο νέφος (cloud) έχει σημειώσει την δημιουργία νέων υπηρεσιών, όπως η Platform as a Service (PaaS), η Infrastructure as a Service (IaaS) και το Software as a Service (SaaS). Η ανάπτυξη των παραπάνω στην αγορά έχει οδηγήσει σε μια μάχη στον κόσμο του cloud computing. Συνδυάζοντας αυτές τις υπηρεσίες που βασίζονται σε cloud δημιουργήθηκε σιγά-σιγά ένας νέος τομέας ανταγωνισμού, η Μηχανική Μάθηση ως υπηρεσία (**Machine Learning as a Service** – MlaaS). Η αυξανόμενη τάση μετατόπισης της αποθήκευσης δεδομένων στο cloud, η διατήρησή τους και η εξαγωγή των καλύτερων στοιχείων από αυτά έχει οδηγήσει στο MLaaS που παρέχει λύσεις μηχανικής μάθησης με μειωμένο κόστος.

Τι είναι το MlaaS

Η μηχανική μάθηση ως υπηρεσία (MLaaS) είναι μια σειρά υπηρεσιών που παρέχουν εργαλεία μηχανικής μάθησης ως μέρος των υπηρεσιών του cloud. Το MLaaS βοηθά τους πελάτες να επωφεληθούν από την μηχανική μάθηση χωρίς το αντίστοιχο κόστος αγοράς εξοπλισμού, χρόνο και κίνδυνο της εγκατάστασης εξοπλισμού μηχανής μάθησης τοπικά. Οι ανησυχίες σχετικά με τις υποδομές, όπως η προεπεξεργασία δεδομένων, η εκπαίδευση μοντέλων, η αξιολόγηση μοντέλων και, τελικά, προβλέψεις, μπορούν να μειωθούν μέσω του MLaaS. Οι πάροχοι υπηρεσιών προσφέρουν εργαλεία όπως analytics και deep learning models, APIs, οπτικοποίηση δεδομένων, επεξεργασία φυσικής γλώσσας και άλλα. Η λειτουργία υπολογισμού διεκπεραιώνεται από το κέντρο δεδομένων (datacenters) του παρόχου υπηρεσιών, απαλλάσσοντας τον πελάτη από τα βάρη εγκατάστασης και συντήρησης των φυσικών μηχανημάτων.

Πώς λειτουργεί το MLaaS

Με απλά λόγια, το MLaaS είναι ένα σύνολο υπηρεσιών που προσφέρουν έτοιμα, ελαφρώς γενικά εργαλεία μηχανικής μάθησης που μπορούν να προσαρμοστούν από οποιονδήποτε οργανισμό ως μέρος των εργασιακών αναγκών τους. Αυτές οι υπηρεσίες κυμαίνονται από την οπτικοποίηση δεδομένων, μέχρι μια σειρά από διεπαφές προγραμματισμού εφαρμογών, αναγνώριση προσώπου, επεξεργασία φυσικής γλώσσας, προγνωστική ανάλυση και βαθιά μάθηση μεταξύ άλλων. Αλγόριθμοι MLaaS χρησιμοποιούνται επίσης για την εύρεση σχεδίων-προτύπων σε δεδομένα. Τα μαθηματικά μοντέλα τους κατασκευάζονται χρησιμοποιώντας μοτίβα από προηγούμενες επεξεργασίες και χρησιμοποιούνται για να κάνουν προβλέψεις νέα δεδομένα σαν είσοδο.

Το κλειδί είναι το γεγονός ότι οι χρήστες (ή πιθανον, οι οργανισμοί που αγοράζουν MLaaS) δεν χρειάζεται να χειριστούν τον πραγματικό υπολογισμό. Τα κέντρα δεδομένων των παρόχων ττον διαχειρίζονται εξ αποστάσεως. Το MLaaS είναι η μόνη πλατφόρμα AI που ενοποιεί συστήματα που κυμαίνονται από κινητές εφαρμογές, επιχειρηματικές πληροφορίες, βιομηχανικό αυτοματισμό και έλεγχο, μέχρι και προηγμένους αισθητήρες όπως το LiDar (LIght Detection And Ranging), μεταξύ άλλων.

Το MLaaS είναι μια πλατφόρμα που παρέχει τόσο αναγνώριση προτύπων όσο και πιθανολογικούς συλλογισμούς. Αυτό προσφέρει μια ευελιξία στη χρήση διαφορετικών μεθόδων

για τη δημιουργία προσαρμοσμένης εργασίας ειδικά για την κάλυψη των αναγκών μιας εταιρείας. Τα MLaaS υποστηρίζονται από αλγόριθμους, όπως συνελικτικά νευρωνικά δίκτυα (CNN), βαθιά νευρωνικά δίκτυα (DNN), Bayesian δίκτυα, πιθανοτικά γραφικά μοντέλα, περιορισμένη μηχανή Boltzmann (RBM) και αναγνώριση προτύπων.

Πολλοί πάροχοι coud, όπως η Microsoft, η Amazon και η IBM, μεταξύ άλλων, προσφέρουν εργαλεία MLaaS. Η μηχανική μάθηση στην αυτοματοποίηση των υπηρεσιών δεν είναι μια νέα έννοια, αλλά το ανανεωμένο ενδιαφέρον για τον τομέα κατά την τελευταία δεκαετία. Η προοδευτική μετάβαση όλων των υπηρεσιών σε clouds καθιστά και το MLaaS ως ένα κατάλληλο εργαλείο του μέλλοντος. Η Amazon με το **Amazon ML**, η Microsoft με το **Azure ML**, το **Watson** της IBM και η Google με το **Google Cloud ML** είναι μερικοί από τους κορυφαίους παρόχους υπηρεσιών MLaaS.

Ακολουθούν μερικές από τις υπηρεσίες MLaaS που προσφέρονται στην αγορά:

- Επεξεργασία φυσικής γλώσσας: Amazon Comprehend, API μοντέλου γλώσσας Azure, API φυσικής γλώσσας του Google Cloud
- Αναγνώριση ομιλίας: Amazon Transcribe, υπηρεσία Azure Custom Speech, Google Dialogflow Enterprise Edition
- Υπολογιστική όραση: Αναγνώριση του Amazon, Υπηρεσία Azure Custom Vision, API Google Cloud Vision
- AI πλατφόρμες: Amazon Sagem aker, Azure Machine Learning Studio, Google Cloud Machine Learning Engine

Το μέλλον του MLaaS

Με τα τρέχοντα δεδομένα και τη δέσμευσή του να λειτρουγεί με cloud τρόπο, το MLaaS θα ενταχθεί στον πυρήνα της μηχανικής μάθησης και θα καταστεί ο πλέον βασικός τρόπος εκπαίδευσης και χρήσης ML στις εφαργμογές. Σύμφωνα με μια μελέτη, η αγορά MLaaS θα σημειώσει αύξηση 49% κατά την περίοδο 2017-2023.

Ένας άλλος τομέας στον οποίο το MLaaS θα μπορούσε να οδηγήσει στην καινοτομία είναι το Διαδίκτυο των Πραγμάτων (**Internet of Things – IoT**). Σύμφωνα με μελέτες, πάνω από 20 δισεκατομμύρια μονάδες εξοπλισμού (εξαιρούνται οι υπολογιστές, τα tablets και τα smartphones) θα αποτελέσουν το IoT μέχρι το 2020. Με το ML να έχει ήδη την ικανότητα να ενσωματώνεται σε διάφορα είδη συσκευών και αισθητήρων, το MLaaS θα μπορούσε να διαδραματίσει βασικό ρόλο σε αυτόν τον τομέα.

ΚΕΦΑΛΑΙΟ 2

2.1 TENSORFLOW

Το Tensorflow είναι μια βιβλιοθήκη λογισμικού ανοιχτού κώδικα που χρησιμοποιείται για αριθμητικούς υπολογισμούς με την χρήση γράφους ροής δεδομένων (data flow graphs). Αναπτύχθηκε από την ομάδα Google Brain μέσα στα πλαίσια της έρευνας της Google, **Machine Intelligence** για την μηχανική μάθηση και την ανάλυση για τα «βαθιά νευρωνικά δίκτυα» (deep neural networks). Παρά την αρχική αιτία δημιουργίας του, το σύστημα πλέον είναι αρκετά γενικευμένο για να εφαρμοστεί σε μια πληθώρα άλλων χρήσεων. Έφτασε στην έκδοση 1.0 τον Φεβρουάριο του 2017 και έχει συνεχίσει με ταχεία ανάπτυξη με περισσότερες από 21,000 αλλαγές κώδικα (commits) μέχρι τώρα, πολλές από εξωτερικούς συνεισφέροντες.

Πρόκειται για ένα εργαλείο cross-platform, και αυτό σημαίνει οτι μπορεί να εκτελεστεί σε CPUs, GPUs – συμπεριλαμβανομένων φορητών συσκευών και ενσωματωμένων συστημάτων, καθώς και TPUs (Tensor Processing Unit – δεν είναι αρκετά διαδεδομένοι ακόμα, αλλά η Google έχει ξεκινήσει ένα σχετικό cloud πρόγραμμα σε έκδοση alpha: https://www.tensorflow.org/tfrc/).

Η εκτέλεση του Tensorflow μπορεί να **κατανέμει** τις διεργασίες της μηχανικής μάθησης σε ποικίλες συσκευές, δημιουργόντας μια ιδιαίτερα υψηλής απόδοσης μηχανή, χάρη στις ανωτέρου επιπέδου διεπαφές σε **Python** και **C++,** «επάνω» από τις οποίες λειτουργούν APIs που είναι διαδεδομένα όπως το **Keras** και το **Estimator API**.

Γράφοι

Η μηχανική μάθηση γίνεται γρήγορα πολύπλοκη, και τα deep learning μοντέλα αρκετά μεγάλα. Για πολλούς γράφους μοντέλων χρειαζόμαστε κατανεμημένη εκπαίδευση για να μπορούμε να τους διαχειριστούμε σε λογικά και αποτελεσματικά χρονικά πλαίσια. Επίσης, υπάρχει η ανάγκη τα μοντέλα να εκπαιδεύονται με τέτοιο τρόπο που να μπορούν να χρησιμοποιηθούν σε πολλές διαφορετικές πλατφόρμες.

Με το Tensorflow γράφουμε κώδικα για να σχηματίσουμε έναν υπολογιστικό γράφο και στην συνέχεια να τον εκτελέσουμε. Ο **γράφος** είναι μια δομή δεδομένων η οποία περιγράφει πλήρως τους υπολογισμούς που θέλουμε να εκτελέσουμε, και έχει μια πληθώρα πλεονεκτημάτων:

- Είναι φορητός, εφόσον μπορεί είτε να εκτελεστεί άμεσα είτε να αποθηκευτεί για μεταγενέστερη χρήση, ενώ μπορεί να λειτουργήσει σε πολλές πλατφόρμες: CPUs, GPUs, TPUs, κινητές και ενσωματωμένες συσκευές. Επίσης, μπορεί να χρησιμοποιηθεί σε στάδια παραγωγής, χωρίς καμία απαίτηση χρήσης του κώδικα που έφτιαξε τον γράφο, απαιτείται απλώς το κομμάτι εκτέλεσης για να χρησιμοποιηθεί.
- Είναι μεταβλητός και βελτιστοποιήσιμος, αφού μπορεί να μεταβληθεί κατάλληλα για να «τρέξει» σε μια διαφορετική πλατφόρμα. Επίσης μπορούν να γίνουν βελτιστοποιήσεις μνήμης και υπολογιστικής δύναμης για να χρησιμοποιηθεί ο ίδιος γράφος σε διαφορετικές πλατφόρμες και συνδυασμούς αυτών. Αυτό είναι χρήσιμο για παράδειγμα στην περίπτωση που χρησιμοποιούμε ένα ή περισσότερα ισχυρά μηχανήματα για να

εκπαιδεύσουμε το μοντέλο και αρκετά χαμηλότερης ισχύος αλλά κινητές πλέον συσκευές για να κάνουν τις προβλέψεις.

- **Υποστηρίζουν κατανεμημένη εκτέλεση**.
  Τα υψηλού επιπέδου APIs του Tensorflow σε συνδυασμό με τους υπολογιστικούς γράφους, δημιουργούν ένα πλούσιο και ευέλικτο περιβάλλον ανάπτυξης και ισχυρές δυνατότητες παραγωγής στο ίδιο πλαίσιο.

Πρόωρη εκτέλεση

Μια πολύ πρόσφατη προσθηκη στο Tensorflow είναι η επιλογή της πρόωρης εκτέλεσης (**Eager execution).** Πρόκειται για μια λειτουργία προγραμματισμού που αξιολογεί τις συναρτήσεις και τις εκτελεί αμέσως, χωρίς να δημιουργεί γράφους. Οι συναρτήσεις επιστρέφουν συγκεκριμένες τιμές αντί να κατασκευάσουν υπολογιστικούς γράφους για να τρέξουν αργότερα. Αυτό καθιστά εύκολο το ξεκίνημα με το Tensorflow και τα μοντέλα εντοπισμού σφαλμάτων (debugging), ενώ μειώνει και τις άσκοπες επαναλήψεις κώδικα (boilerplate). Οι πιο βασικοί λόγοι χρήσεις της πρόωρης εκτέλεσης είναι:

- Γρηγορότερος έλεγχος κώδικα και του γράφου
- Χρήση του ελέγχου ροής της Python μέσα από τα Tensorflow APIs (επαναλήψεις, συνθήκες, συναρτήσεις κτλ)
- Πιο άμεση αποσφαλμάτωση
- Η σημασία της εκχώρησης κατα την εκτέλεση (της πρόωρης εκτέλεσης) καθιστά εύκολη την κατασκευή δυναμικών γράφων

Όταν ο προγραμματιστής είναι ικανοποιημένος από την πρόωρη εκτέλεση, μπορεί να μετατρέψει αυτομάτως τον κώδικά του σε γράφο για να αποθηκευτεί και να χρησιμοποιηθεί σε άλλη πλατφόρμα ή σε μεταγενέστερη εκτέλεση.

2.2 ΚΑΤΑΝΕΜΗΜΕΝΗ ΕΚΠΑΙΔΕΥΣΗ

Για την παράλληλη εκπαίδευση, χρησιμοποιούνται δυο διαφορετικές προσεγγίσεις, που ονομάζονται «**παραλληλισμός δεδομένων**» και «**παραλληλισμός μοντέλου**». Στον παραλληλισμό των δεδομένων χρησιμοποιείται το ίδιο μοντέλο για κάθε νήμα (thread), αλλά τροφοδοτείται με διαφορετικά κομμάτια (batches) δεδομένων. Στον παραλληλισμό μοντέλου χρησιμοποιούνται τα ίδια δεδομένα για κάθε νήμα, αλλά διαμοιράζουμε τις διαφορετικές λειτουργίες του μοντέλου μεταξύ των νημάτων.

Στα νευρωνικά δίκτυα αυτό σημαίνει ότι ο παραλληλισμός των δεδομένων χρησιμοποιεί τα ίδια βάρη και διαφορετικές μίνι-παρτίδες δεδομένων σε κάθε νήμα. Οι κλίσεις (gradients) πρέπει να συγχρονιστούν, δηλ. να υπολογιστεί ο μέσος όρος τους, μετά από κάθε πέρασμα μιας μίνι-παρτίδας δεδομένων. Ο παραλληλισμός μοντέλου χωρίζει τα βάρη του δικτύου σε ίσα μέρη μεταξύ των νημάτων και όλα τα νήματα λειτουργούν με την ίδια μίνι-παρτίδα δεδομένων την φορά. Σε αυτή την περίπτωση η παραγόμενη έξοδος μετά από κάθε επίπεδο πρέπει να συγχρονιστεί, δηλαδή να αθροίζεται, για να παρέχει την είσοδο στο επόμενο επίπεδο.

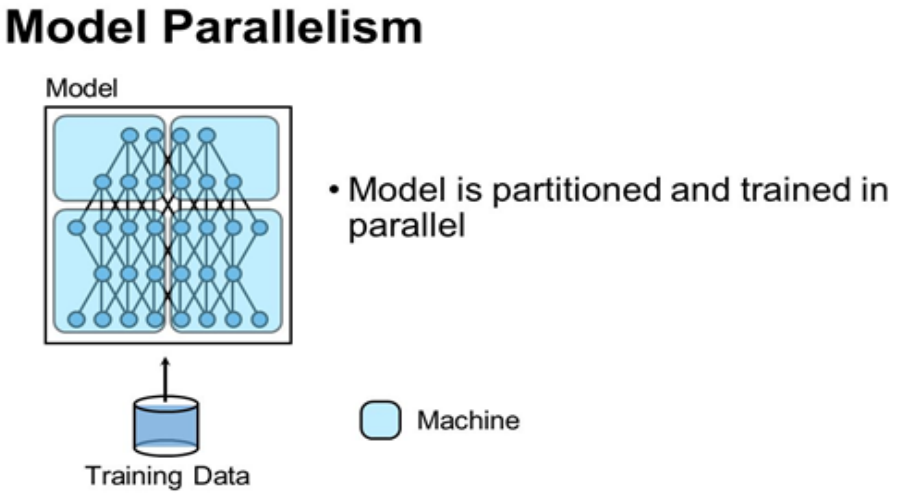
Κάθε μέθοδος έχει τα πλεονεκτήματα και τα μειονεκτήματά της που αλλάζουν από αρχιτεκτονική σε αρχιτεκτονική. Ας εξετάσουμε πρώτα τον τον παραλληλισμό του μοντέλου και στην συνέχεια θα αναλύσουμε τον παραλληλισμό των δεδομένων και τα σημεία συμφόρησης.

Παραλληλισμός μοντέλου

Παραλληλισμός μοντέλου ονομάζεται η προσέγγιση παράλληλης εκπαίδευσης, κατά την οποία διαιρείται το μοντέλο, διαμοιράζεται μεταξύ των μονάδων επεξεργασίας (πχ GPU), και δίνουμε σαν είσοδο τα ίδια δεδομένα για κάθε (αντίγραφο) μοντέλο. Στην μηχανική μάθηση, ένας τρόπος για να γίνει αυτό, είναι με τη διάσπαση των βαρών, π.χ. μια μήτρα βάρους 1000×1000 θα χωριστεί σε μικρότερες μήτρες 1000×250 αν χρησιμοποιήσουμε τέσσερις GPU.

Ένα πλεονέκτημα αυτής της προσέγγισης είναι προφανές από την αρχή: εάν χωρίσουμε τα βάρη μεταξύ των GPUs, μπορούμε να έχουμε πολύ μεγάλα νευρωνικά δίκτυα των οποίων τα βάρη δεν θα χρειάζεται να χωρέσουν στην μνήμη μιας GPU. Βέβαια κατά κύριο λόγο, τέτοιου μεγάλου μεγέθους νευρωνικά δίκτυα είναι συνήθως περιττά. Ωστόσο, για πολύ μεγάλες μη επιτηρούμενες διαδικασίες μάθησης - οι οποίες θα γίνουν πολύ σημαντικές στο κοντινό μέλλον - τέτοια μεγάλα δίκτυα θα χρειαστούν για να εκπαιδεύσουν πιο λεπτά χαρακτηριστικά, που θα μπορούσαν να οδηγήσουν σε ακόμα πιο «έξυπνη» συμπεριφορά.

Χρησιμοποιώντας στην πράξη την λογική αυτή, πρέπει να συγχρονίσουμε (προσθέτοντας ή αθροίζοντας) τα βάρη μετά από κάθε πέρασμα και αυτό φαίνεται πιο αργό σε σύγκριση με τον παραλληλισμό των δεδομένων, όπου συγχρονίζουμε μόνο μία φορά. Η ουσιώδης διαφορά είναι οτι στον παραλληλισμό δεδομένων, ένα gradient 1000x500 πρέπει να μεταφερθεί μία φορά για το κάθε επίπεδο 1000×500 - δηλαδή 500.000 στοιχεία. Στον παραλληλισμό μοντέλου πρέπει απλά να μεταφέρουμε μια μικρή μήτρα για κάθε προς τα εμπρός και προς τα πίσω πέρασμα, με ένα σύνολο 128.000 ή 160.000 στοιχείων - δηλαδή σχεδόν 4 φορές λιγότερα δεδομένα. Έτσι, το εύρος ζώνης της κάρτας δικτύου είναι το κύριο εμπόδιο σε ολόκληρη την εφαρμογή, αλλά επηρεάζει την κατάσταση πολύ λιγότερο από ό,τι στην περίπτωση του παραλληλισμού δεδομένων.

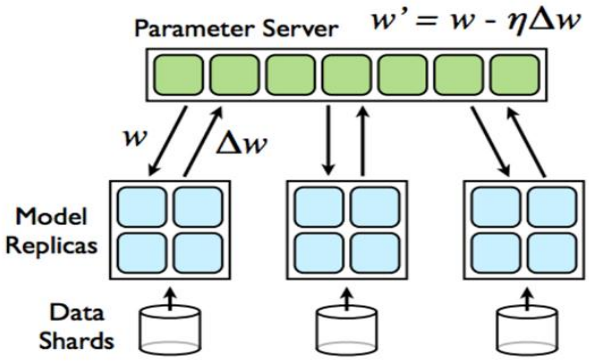


Εικόνα 10: Απλός παραλληλισμός μοντέλου

Παραλληλισμός δεδομένων

Η ιδέα του παραλληλισμού των δεδομένων είναι λίγο πιο απλή. Εάν έχουμε για παράδειγμα στην διάθεσή μας 4 μονάδες GPU, χωρίζουμε ένα batch δεδομένων σε 4 μέρη, ένα για κάθε μια GPU. Για το συγκεκριμένο παράδειγμα με τις 4 επεξεργαστικές μονάδες, χωρίζουμε ένα batch με 128 παραδείγματα σε 32 παραδείγματα για κάθε GPU. Στη συνέχεια, τροφοδοτούμε το αντίστοιχο batch στο δίκτυο και παίρνουμε τα gradients για κάθε ένα από τα 4 κομμάτια του batch (32+32+32+32 = 128). Έπειτα, είτε με την χρήση του MPI, είτε με λειτουργίες που προσφέρει το πλαίσιο (framework) με το οποίο εργαζόμαστε (πχ Tensorflow) συλλέγουμε όλα τα gradients και ενημερώνουμε τις παραμέτρους με το συνολικό μέσο όρο.

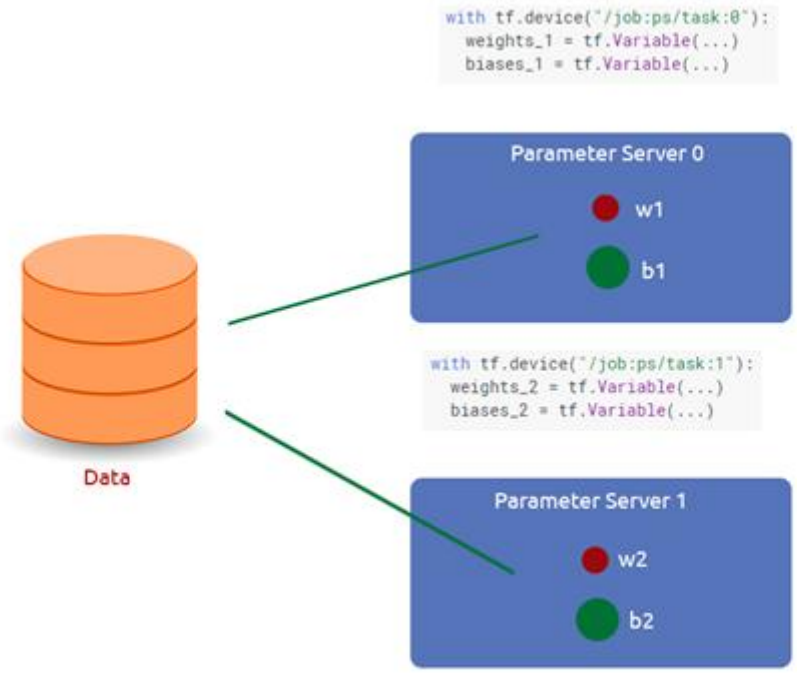
Το μεγαλύτερο πρόβλημα με αυτήν την προσέγγιση είναι ότι κατά τη διάρκεια του περάσματος προς τα πίσω, πρέπει να περάσουμε ολόκληρα gradients σε όλες τις άλλες μονάδες GPU. Αν για παράδειγμα έχουμε matrix 1000×1000 τότε θα πρέπει να περάσουν 4.000.000 byte σε κάθε δίκτυο. Αν χρησιμοποιήσουμε μια κάρτα δικτύου 40Gbit/s - η οποία είναι αρκετά γρήγορη - τότε θα χρειαστούμε $\frac{4000000}{40}\frac{1}{40\times1024\times1024\times102}\frac{1}{8\times1000} = 0.75\text{ms}$ για να μεταβιβάσουμε τα δεδομένα από έναν κόμβο σε έναν άλλο κόμβο (να σημειωθεί ωστοσο οτι στο δίκτυο υπάρχουν κάποιες επιπλέον καθυστερήσεις που παραμελήσαμε εδώ). Εάν έχουμε 6 GPU σε δύο μηχανήματα, πρέπει να μεταδώσουμε τα δεδομένα σε άλλες 5 GPU, τρεις από τις οποίες πρέπει να περάσουν από την κάρτα δικτύου (3 x 0,75ms), ενώ δύο μπορούν να χρησιμοποιήσουν τον PCIe για να μεταδώσουν τα δεδομένα στις άλλες δύο μονάδες GPU (περίπου τρεις φορές πιο γρήγορα, 2 φορές 0,25ms). Ωστόσο, το πέρασμα PCIe είναι ανεξάρτητο από την λειτουργία της κάρτας δικτύου, οπότε ο απαιτούμενος χρόνος καθορίζεται από τον χρόνο της κάρτας δικτύου μόνο, δηλαδή 2,25ms. Όμως μόνο μία GPU μπορεί να μεταφέρει δεδομένα μέσω της κάρτας δικτύου σε οποιαδήποτε στιγμή σε οποιονδήποτε μηχάνημα, οπότε χρειάζεται να πολλαπλασιάσουμε αυτόν τον χρόνο επι 3, δηλ. 7,75ms. Η ελάχιστη απαίτηση είναι ότι χρειαζόμαστε μόνο περίπου 0,2ms για μια μήτρα που πολλαπλασιάζεται μέσω αυτού του στρώματος (100 × 1000 . 1000 × 1000) και περίπου διπλάσια για το πίσω πέρασμα. Μπορούμε να περάσουμε gradients ενώ δουλεύουμε στο επόμενο επίπεδο, αλλά τελικά η ταχύτητα της κάρτας δικτύου περιορίζει τον συνολικό μας υπολογισμό αρκετά. Αυτό γίνεταα περισσότερο εμφανές όσο μεγαλώνει ο αριθμός των μηχανών στο δίκτυο (στο ίδιο cluster): Ένα σύστημα τεσσάρων κόμβων που εργάζεται για το ίδιο πρόβλημα χρειάζεται περίπου 20,25ms για να περάσει τις κλίσεις στις υπόλοιπες GPU. Μπορούμε εύκολα να δούμε ότι ο παραλληλισμός των δεδομένων δεν κλιμακώνεται με το μέγεθος του cluster.



Εικόνα 11: Παραλληλισμός δεδομένων

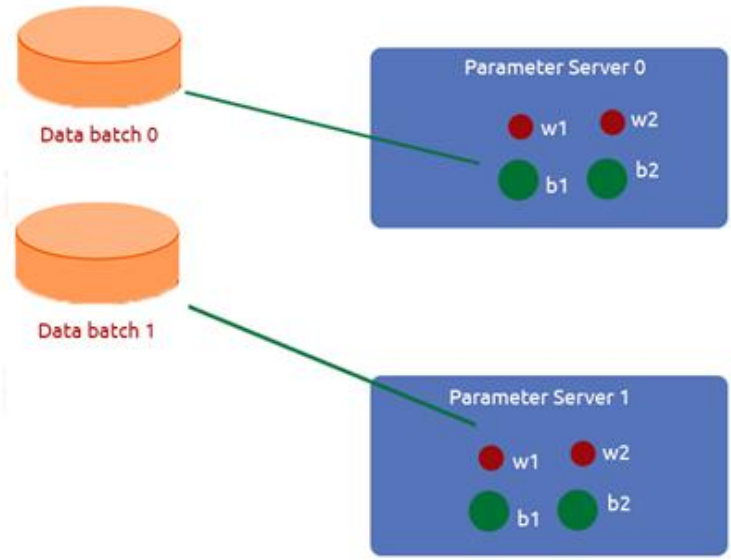
Σύγκριση παραλληλισμού δεδομένων / μοντέλου

Για να συνοψίσουμε και να συγκρίνουμε τις δυο προσεγγίσεις, ο βασικός διαχωρισμός αφορά τις λειτουργίες που θα εκτελέσει κάθε συσκευή (ονομάζονται **workers** στο documentation του Tensorflow) και τα δεδομένα στα οποία θα τις εκτελέσει. Κατα τον παραλληλισμό μοντέλου, χωρίζουμε τις λειτουργίες του μοντέλου σε διαφορετικές συσκευές (workers) και τα τροφοδοτούμε ίδια δεδομένα. Έτσι, διαφορετικά επίπεδα σε ένα δίκτυο μπορούν να εκπαιδευτούν παράλληλα σε διαφορετικές μονάδες GPU. Για παράδειγμα, οι υπολογισμοί weights_1 & biases_1 εκτελούνται στο μηχάνημα 0, ενώ οι υπολογισμοί weights_2 & biases_2 εκτελούνται στο μηχάνημα 1. Αυτή η διαδικασία ονομάζεται για το Tensorflow **In-graph replication**. Με αυτή την προσέγγιση είναι δύσκολο να έχουμε πετύχουμε δυνατές επιδόσεις, ειδικά σε μικρά μοντέλα.



Εικόνα 12: Κατανομή μοντέλου σε μηχανές

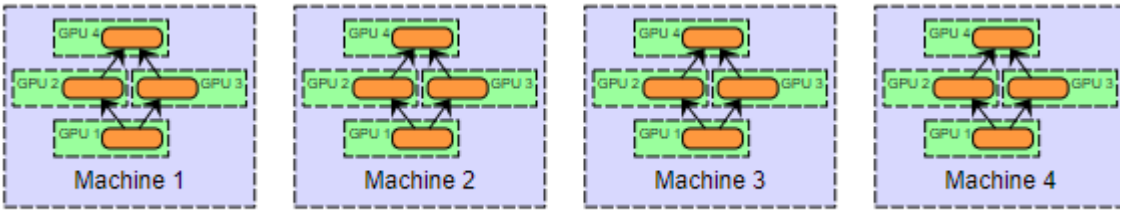
Στον παραλληλισμό δεδομένων (ή **Between-graph replication** στην περίπτωση του Tensorflow), χρησιμοποιούμε το ίδιο μοντέλο για κάθε συσκευή, αλλά το εκπαιδεύουμε χρησιμοποιώντας διαφορετικά τμήματα δεδομένων εκπαίδευσης σε κάθε συσκευή. Αυτό έρχεται σε αντίθεση με τον παραλληλισμό μοντέλο, ο οποίος χρησιμοποιεί τα ίδια δεδομένα για κάθε συσκευή, αλλά χωρίζει το μοντέλο από τις συσκευές. Κάθε συσκευή θα υπολογίσει ανεξάρτητα τα σφάλματα μεταξύ των προβλέψεών της για τα δείγματα εκπαίδευσης και τις ετικέτες εξόδων (σωστές τιμές για αυτά τα δείγματα εκπαίδευσης). Επειδή κάθε συσκευή εκπαιδεύει σε διαφορετικά δείγματα, υπολογίζει διαφορετικές αλλαγές στο μοντέλο (gradients). Ωστόσο, ο αλγόριθμος εξαρτάται από τη χρήση των συνδυασμένων αποτελεσμάτων όλων των επεξεργασιών για κάθε νέα επανάληψη, ακριβώς όπως και αν ο αλγόριθμος έτρεχε σε ένα μόνο μηχάνημα (πχ μια μεμονωμένη CPU). Επομένως, κάθε συσκευή πρέπει να στείλει όλες τις αλλαγές σε όλα τα μοντέλα σε όλες τις άλλες συσκευές.



Εικόνα 13: Κατανομή δεδομένων σε μηχανές με το ίδιο μοντέλο

Ενώ ο παραλληλισμός μοντέλου μπορεί να λειτουργήσει καλά στην πράξη, ο παραλληλισμός δεδομένων είναι αναμφίβολα η προτιμώμενη προσέγγιση για τα κατανεμημένα συστήματα και αποτέλεσε το επίκεντρο περισσότερης έρευνας. Γενικά, η εφαρμογή, η ανοχή σφάλματος και η καλή χρήση του συνόλου των μονάδων επεξεργασίας είναι ευκολότερη στον παραλληλισμό δεδομένων παρά στον παραλληλισμό του μοντέλου. Ο παραλληλισμός μοντέλου στο πλαίσιο των κατανεμημένων συστημάτων είναι ενδιαφέρων και έχει ορισμένα οφέλη (όπως η δυνατότητα κλιμάκωσης σε μεγάλα μοντέλα), αλλά στην παρούσα πτυχιακή εργασία θα επικεντρωθούμε στον παραλληλισμό των δεδομένων.

Φυσικά, οι προσεγγίσεις δεν αποκλείουν η μια την άλλη. Υπάρχουν περιπτώσεις που χρησιμοποιούνται ομάδες συστημάτων πολλαπλών GPU. Θα μπορούσαμε να χρησιμοποιήσουμε παράλληλο μοντέλο (μοντέλο χωρισμένο σε GPUs) για κάθε μηχανή και παραλληλισμό δεδομένων μεταξύ μηχανών.



Εικόνα 14: Συνδυασμός παραλληλισμού δεδομένων και παραλληλισμού μοντέλου

Κατανεμημένη εκπαίδευση με Tenrosflow

Το Tensorflow διατηρεί την δομή του cluster σε ένα αντικείμενο τύπου `tf.train.ClusterSpec` από το οποίο γνωρίζει η κάθε μηχανή την ύπαρξη των υπολοίπων και συντονίζεται η λειτουργία

τους. Το αντικείμενο ClusterSpec περιέχει αντικείμενα του τύπου `tf.train.Server`, τα οποία αναπαριστούν σύνολα συσκευών (`tf.Device`) και συνεδρίες (`tf.Session`) που συμμετέχουν στην διαδικασία εκπαίδευσης. Κάθε `tf.train.Server` αντιστοιχεί σε ένα «καθήκον» (task) που ανήκει σε μια εργασία (job).

Για παράδειγμα:

```
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
server = tf.train.Server(cluster,
                         job_name=worker,
                         task_index=0)
```

Η δεύτερη γραμμή, στην οποία δημιουργείται ο server, δημιουργείται και ο $worker_0$, στον οποίο θα ανατεθεί εργασία με την

```
with tf.device("/job:%s/task:%d" % (job_name, task_index)):
```

όπου job_name = worker και task_index = 0.

*Σημείωση: οι παράμετροι ps_hosts και worker_hosts είναι πίνακες που διατηρούν τις διευθύνσεις των αντίστοιχων servers. Το αντικείμενο worker_hosts θα μπορούσε να έχει την τιμή ["192.168.10.7:2222", "192.168.10.8:2222", "192.168.10.9:2222"] για διαφορετικές μηχανές στο cluster, ή ["192.168.10.7:2222", "192.168.10.7:2223", "192.168.10.7:2224"] για πολλές διεργασίες worker στην ίδια μηχανή.

Τόσο οι workers όσο και οι ps αντιμετωπίζονται από το Tensorflow ως «καθήκοντα» (tasks – για την ακρίβεια «εργασίες» (jobs), που είναι απλά ομάδες καθηκόντων), έτσι έχουν πολλά κοινά στην δομή τους. Η διαφορά τους έγκυται στην ιδιαιτερότητα του σκοπού χρήσης τους. Η ιδέα είναι ότι η κατάσταση των παραμέτρων (π.χ. `tf.Variable`) πρέπει να βρίσκεται στους ps, ενώ οι λειτουργίες για τον υπολογισμό της κατάστασης του γράφου θα πρέπει να εκτελούνται στους workers. Το Tensorflow αντί να το επιτύχει καλώντας το αντικείμενο `tf.device` – το οποίο αντιπροσωπεύει μια μηχανή στο cluster – χειροκίνητα παντού, χρησιμοποιείται μια βοηθητική λειτουργία με όνομα `tf.train.replica_device_setter` που αναθέτει την αποθήκευση των παραμέτρων `tf.Variable` σε ένα ps και τις άλλες λειτουργίες σε έναν worker.

```
cluster_spec = {
       "ps": ["ps0:2222", "ps1:2222"],
       "worker": ["worker0:2222", "worker1:2222", "worker2:2222"]}
with tf.device( tf.train.replica_device_setter( cluster = cluster_spec )):


# Build graph
  v1 = tf.Variable(...)  # assigned to /job:ps/task:0
  v2 = tf.Variable(...)  # assigned to /job:ps/task:1
  v3 = tf.Variable(...)  # assigned to /job:ps/task:0
  # Run computations with workers 0, 1, 2
```

Να σημειωθεί οτι μέσα στους ps που ανήκουν σε κάποιο cluster, είναι απαραίτητη η χρήση της `server.join()`, που σημαίνει απλά ότι οι διακομιστές παραμέτρων θα περιμένουν τους workers, αντί να τερματίσουν αμέσως τις διαδικασίες τους.

Σύγχρονη / Ασύγχρονη εκπαίδευση

Στην διαδικασία του παραλληλισμού δεδομένων, υπάρχουν βασικές υποκατηγορίες:

- Ασύγχρονη εκπαίδευση (Asynchronous training): Σε αυτή την περίπτωση κάθε worker διαβάζει τις παραμέτρους, υπολογίζει ενημερώσεις και γράφει ενημερωμένες παραμέτρους, χωρίς κανένα μηχανισμό κλειδώματος. Σε ένα πιθανό σενάριο, ο worker 1 είναι αργός για κάποιο λόγο. Ο εργαζόμενος 1 διαβάζει τις παραμέτρους σε χρόνο t και στη συνέχεια προσπαθεί να γράψει ενημερωμένες παραμέτρους στον χρόνο t + 100. Εν τω μεταξύ, οι workers 2-n έχουν κάνει πολλές ενημερώσεις στα χρονικά σημεία t + 1, t + 2 κλπ. Όταν ο αργός worker 1 κάνει τελικά την ενημέρωσή του, αυτή συνυπολογίζεται με όλη την πρόοδο που έχουν κάνει οι άλλοι εργαζόμενοι.
- Σύγχρονη εκπαίδευση (Synchronous training): Πλήρως σύγχρονη (fully synchronous) σημαίνει ότι όλοι οι workers συντονίζονται. Κάθε worker διαβάζει τις παραμέτρους, υπολογίζει μια κλίση (gradient) και στη συνέχεια περιμένει τους υπόλοιπους να τελειώσουν. Στη συνέχεια, ο αλγόριθμος μάθησης υπολογίζει το μέσο όρο όλων των διαβαθμίσεων που υπολογίζουν και κάνει μια ενημέρωση βασισμένη σε αυτόν τον μέσο όρο. Εάν ο worker 1 είναι πολύ αργός και παίρνει 100 χρονικά σημεία για να τελειώσει, αλλά οι εργαζόμενοι 2-n τελειώνουν στο χρονικό σημείο 2, τότε οι περισσότεροι workers θα περάσουν το μεγαλύτερο μέρος του χρόνου άπραγοι ενώ θα περιμένουν τον πρώτο worker να ολοκληρώσει.

Όταν εκπαιδεύουμε ασύγχρονα ένα μοντέλο στο Tensorflow, ένας συγκεκριμένος worker κάνει τα εξής:

1. Ο worker διαβάζει όλες τις παραμέτρους του κοινόχρηστου μοντέλου παράλληλα από τον κατάλληλο PS και τις αντιγράφει τοπικά. Αυτές οι αναγνώσεις είναι ασύγχρονες και ανεξάρτητες από τυχόν ταυτόχρονες εγγραφές, ενώ δεν χρησιμοποιείται μηχανισμός κλειδώματος: συγκεκριμένα ο εργαζόμενος μπορεί να βλέπει μερικές ενημερώσεις από έναν ή περισσότερους άλλους εργαζόμενους (π.χ. ένα υποσύνολο ενημερώσεων από άλλον εργαζόμενο μπορεί να έχει εφαρμοστεί ή ένα υποσύνολο των στοιχείων σε μια μεταβλητή μπορεί να έχουν ενημερωθεί).

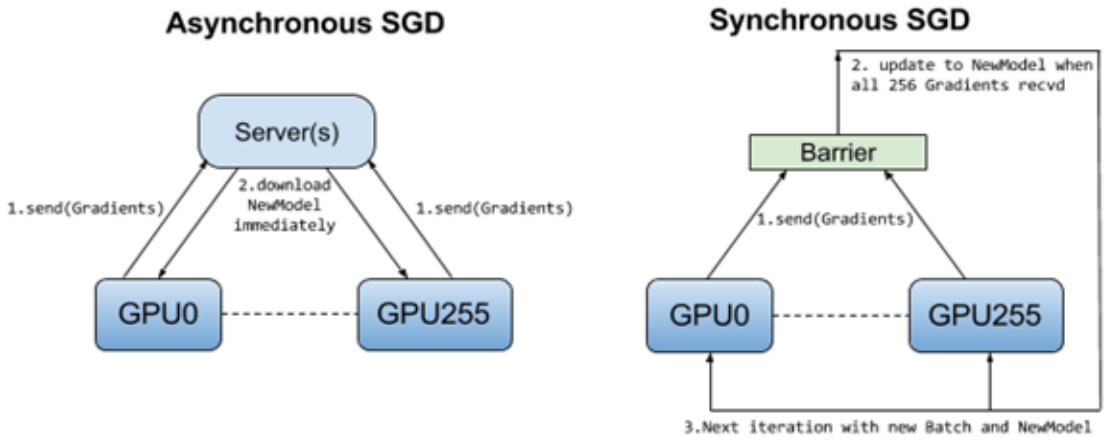
2. Ο worker υπολογίζει τις κλίσεις (gradients) τοπικά, με βάση μια παρτίδα δεδομένων εισόδου και τις τιμές παραμέτρων που διαβάζει στο βήμα 1.

3. Στέλνει τις κλίσεις για κάθε μεταβλητή στην κατάλληλη διεργασία PS και εφαρμόζει τις διαβαθμίσεις στην αντίστοιχη μεταβλητή τους, χρησιμοποιώντας έναν κανόνα ενημέρωσης που καθορίζεται από τον αλγόριθμο βελτιστοποίησης (π.χ. SGD, Adagrad, Adam κ.λπ.). Οι κανόνες ενημέρωσης χρησιμοποιούν συνήθως commutative operations, έτσι ώστε να μπορούν να εφαρμοστούν ανεξάρτητα στις ενημερώσεις από κάθε εργαζόμενο και η κατάσταση κάθε μεταβλητής θα είναι ένα τρέχον σύνολο των ακολουθιών των ενημερωμένων εκδόσεων.

Στην ασύγχρονη εκπαίδευση, κάθε ενημερωμένη έκδοση από έναν worker εφαρμόζεται ταυτόχρονα στον ps και οι ενημερώσεις μπορεί να είναι κάπως συντονισμένες εάν η προαιρετική

use_locking = True σημαία (flag) είχε οριστεί όταν ξεκίνησε η αντίστοιχη βελτιστοποίηση (π.χ. tf.train.GradientDescentOptimizer(use_locking=True)). Σημειώνουμε ωστόσο, ότι το κλείδωμα εδώ παρέχει μόνο αμοιβαίο αποκλεισμό για δύο ταυτόχρονες εγγραφές (ενημερώσεις) και δεν λειτουργεί ως μηχανισμός κλειδώματος για ανάγνωση. Το κλείδωμα δεν παρέχει ατομικότητα σε ολόκληρο το σύνολο των ενημερώσεων.

Αντιθέτως, στην σύγχρονη εκπαίδευση, η χρήση ενός βοηθήματος όπως το `tf.train.SyncReplicasOptimizer` θα διασφαλίσει ότι όλοι οι workers διαβάζουν τις ίδιες, ενημερωμένες τιμές για κάθε παράμετρο του μοντέλου, και ότι όλες οι ενημερώσεις για ένα συγχρονισμένο βήμα συγκεντρώνονται και εφαρμόζονται οι μέσοι όροι πριν εφαρμοστούν στις υποκείμενες μεταβλητές. Για να γίνει αυτό, οι εργαζόμενοι συγχρονίζονται από ένα φράγμα (barrier), το οποίο σταματούν μετά την αποστολή της ενημερωμένης έκδοσης κλίσης τους και «περιμένουν» την εφαρμογή της συγκεντρωτικής ενημέρωσης σε όλες τις μεταβλητές και από όλους τους workers.



Εικόνα 15: Σύγχρονη και ασύγχρονη εκπαίδευση

Παραλληλισμός στο Tensorflow

- **In-Graph replication**: Σε ένα από τα μηχανήματα (parameter server) δημιουργείται ο γράφος με την μορφή ενός αντικειμένου της κλάσης `tf.Graph`, που κρατάει από ένα αντικείμενο τύπου `tf.Variable` για τις παραμέτρους – η καθεμία από τις οποίες κατανέμεται στον ή στους σέρβερ παραμέτρων(/job:ps) και πολλαπλά αντίτυπα των υπολογιστικών κομματιών του μοντέλου τα οποία κατανέμονται ξεχωριστά σαν διεργασίες (tasks) στο/α μηχάνημα/τα (/job:worker). Έτσι κάθε μηχάνημα που εκπαιδεύει το μοντέλο, θα πρέπει να κρατάει αντίγραφο της συνολικής προόδου και να ενημερώνει – ενημερώνεται για αυτήν από τον parameter server. Επειδή στο in-graph replication όλη η ροή του training ελέγχεται από μία συνεδρία – αντικείμενο της κλάσης `tf.Session` που τρέχει στον master worker – και το οποίο είναι υπεύθυνο και για την διαχείριση των υπόλοιπων task στους άλλους workers, αυτό έχει το αποτέλεσμα να δημιουργείται “bottleneck”(συμφόρηση των διεργασιών) σε μεγάλης έκτασης δίκτυα με πολλά αντίτυπα κομματιών (nodes) του γράφου. Η μόνη περίπτωση στην οποία ακόμα χρησιμοποιείται in-

graph replication είναι για την χρήση πολλαπλών συσκευών(CPUs,GPUs,TPUs) σε ένα process/client όπως γίνεται στο παράδειγμα του Tensorflow CIFAR-10 with multiple GPUs.

- **Between-Graph replication**: Στην εκδοχή αυτή υπάρχουν πολλαπλοί clients (διαφορετικές εκτελέσεις του ίδιου προγράμματος σε διαφορετικές μηχανές ή διαφορετικές διεργασίες – στην πράξη πολλά αντίγραφα του προγράμματος που εκτελούνται ταυτόχρονα) κάθε ένας worker ελέγχεται από έναν client δηλαδή ένα ξεχωριστό αντικείμενο `tf.Session` και στον καθένα δημιουργείται ένας παρόμοιος tf.Graph, που δεν είναι αναγκαστικά ίδιος με τους υπόλοιπους. Συχνά χρησιμοποιείται ένας global default γράφος ο οποίος αντιγράφεται από τους clients μέσω tf.get_default_graph() με τους παραμέτρους να κατανέμονται στους/ον ανάλογους/γο parameter server /job:ps. Επίσης χρησιμοποιείται η tf.train.replica_device_setter όπως αναφέραμε νωρίτερα, ώστε οι παράμετροι να καταχωρούνται σε συγκεκριμένα devices και όπου δημιουργούνται παράμετροι με ίδια ονόματα σε διαφορετικά devices(CPUs,GPUs,TPUs) να συγχωνεύονται και να σώζονται σε μια εμφάνιση (instance) στον ps. Επιπρόσθετα σε κάθε replica (αντίγραφο) του γράφου υπάρχει και αντίτυπο των υπολογιστικών μονάδων(nodes) του μοντέλου.

Το μειονέκτημα της between-graph εκδοχής είναι ότι δυσκολεύει αρκετά την κατάσταση στην σύγχρονη εκπαίδευση, επειδή πρέπει να συγχρονιστούν πολλαπλά training loops και να ενημερώνουν την τιμή σε ένα σημειο. Αντίθετα με την in-graph εκδοχή χρησιμοποιείται ένα μοναδικό training loop που εκτελείται σε μία διεργασία. Σε αυτή την περίπτωση κάθε worker έχει μόνο ένα αντίγραφο του γράφου κι ένα κομμάτι (batch) δεδομένων που εκπαιδεύεται τοπικά.

Για να επιτευχθεί αυτό, το Tensorflow προσφέρει την μέθοδο `tf.train.get_or_create_global_step()`. Ο όρος global_step αφορά το πλήθος των κομματιών δεδομένων (batches) στα οποία έχει πρόσβαση ο γράφος. Κάθε φορά που παρέχεται ένα batch, τα βάρη ενημερώνονται προς την κατεύθυνση που ελαχιστοποιεί την απώλεια. Το global_step παρακολουθεί μόνο τον αριθμό των κομματιών που εμφανίστηκαν μέχρι την εκάστοτε χρονική στιγμή. Όταν ο γράφος στην παράμετρο της get_or_create_global_step είναι ίσος με None (Graph=None) τότε η μεταβλητή δημιουργείται και αρχικοποιείται, ενώ όταν η εκτέλεση κάθε client προχωράει στην minimize(), η μεταβλητή αυξάνεται κατά ένα.

Όλα τα παραπάνω καθιστούν την between-graph εκδοχή πιο αποτελεσματική για ασύγχρονη εκπαίδευση, αφού εκτελούνται ταυτόχρονα πολλές επαναλήψεις (loops) που πρέπει να συγχρονιστούν – απαίτηση δύσκολη για την σύγχρονη εκπαίδευση (bottleneck).

Η σύγχρονη εκτέλεση, προτιμάται συνήθως σε περιπτώσεις μεγάλων υπολογιστών πολλαπλών CPU ή αρχιτεκτονικών παράλληλης επεξεργασίας και VLSI chips, ενώ ασύγχρονη εκτέλεση συναντάμε συνήθως σε συμβατικούς υπολογιστές με μερικούς πυρήνες σε CPU & GPU, μικρά δίκτυα υπολογιστών και μηχανήματα με μια ή λίγες μονάδες GPU.

Συνεδρία και επίβλεψη

Ένα αντικείμενο Session περικλείει το περιβάλλον στο οποίο εκτελούνται οι λειτουργίες (operation) και αξιολογούνται αντικείμενα Tensor. Για παράδειγμα:

```
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
sess = tf.Session()
# Υπολογισμός και εκτύπωση του tensor `c`.
print(sess.run(c))
```

Σε ενα session ανήκουν αντικείμενα κλάσεων όπως τα tf.Variable που αναφέραμε νωρίτερα. Είναι σημαντικό να απελευθερώνουμε το σύστημα (την μνήμη) όταν δεν είναι πλέον απαραίτητα. Για τον λόγο αυτό είτε χρησιμοποιούμε την tf.Session.run για να εκτελέσουμε την συνεδρία και στην συνέχεια τερματίζουμε με την tf.Session.close, είτε χρησιμοποιούμε την συνεδρία με την βοήθεια της `with` σαν `context manager`:

```
# .run() και .close().
sess = tf.Session()
sess.run(...)
sess.close()
# context manager.
with tf.Session() as sess:
   sess.run(...)
```

Στην έκδοση 1.8 του Tensorflow, την τελευταία διαθέσιμη αυτή την στιγμή, χρησιμοποιείται η κλάση `tf.train.MonitoredSession`. Πρόκειται για ένα αντικείμενο παρόμοιο με την Session το οποίο χειρίζεται την αρχικοποίηση, την επαναφορά (συνέχιση σε περίπτωση διακοπής) και την εκτέλεση του γράφου. Χρησιμοποιείται όπως στο παρακάτω παράδειγμα:

```
with MonitoredSession(session_creator=ChiefSessionCreator(...),
                      hooks=[saver_hook]) as sess:
  while not sess.should_stop():
      sess.run(train_op)
```

Η παράμετρος saver_hook είναι αντικείμενο της κλάσης tf.train.CheckpointSaverHook το οποίο αποθηκεύει checkpoints (ενδιάμεσα στάδια εκπαίδευσης του μοντέλου, αποθηκευμένα σε μορφή εξαρτημένη από τον κώδικα που δημιούργησε το μοντέλο – σε αντίθεση με το αντικείμενο SavedModel που είναι μορφή του μοντέλου ανεξάρτητη του κώδικα που το δημιούργησε) κάθε n λεπτά ή βήματα εκπαίδευσης. Η μέθοδος MonitoredSession.should_stop() επιστρέφει μια Boolean τιμή, αληθή στην περίπτωση που κάποιος server αδυνατεί να συνεχίσει την εκτέλεση του μοντέλου. Σε αυτή την περίπτωση η εκτέλεση θα συνεχιστεί από το τελευταίο στάδιο στο οποίο είχε αποθηκευτεί checkpoint. Σε διαφορετική περίπτωση η εκτέλεση συνεχίζεται για άλλο ένα βήμα (sess.run()).

Σε παλαιότερες εκδόσεις αυτλα τα καθήκοντα είχε η κλάση tf.train.Supervisor, ένας «βοηθός» της εκπαιδευτικής διαδικασίας που αποθηκεύει checkpoints και υπολογίζει μέσους όρους. Η χρήση του γίνεται με την tf.train.Supervisor.managed_session() στο ακόλουθο παράδειγμα:

```
sv = Supervisor(logdir='/tmp/mydir')
with sv.managed_session(FLAGS.master) as sess:
    while not sv.should_stop():
    sess.run(…)
```

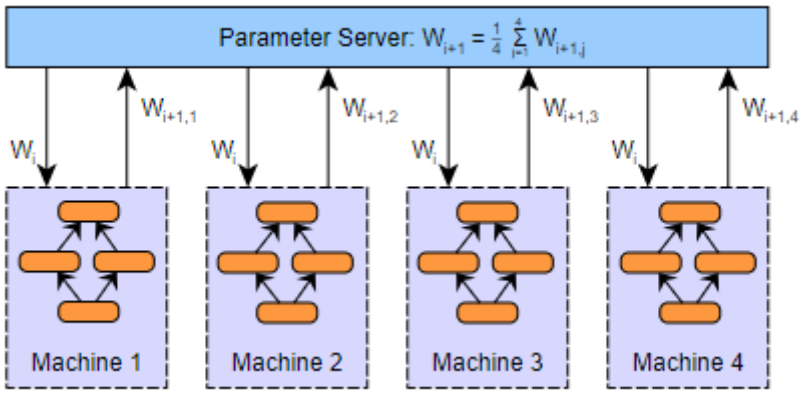
Μέσα σε ένα Session, χρησιμοποιείται η tf.name_scope για να ορίσει το περιβάλλον μέσα στο οποίο ανήκει ένα αντικείμενο, να ομαδοποιήσει δηλαδή αντικείμενα. Η name_scope φαίνεται ιδιαίτερα χρήσιμη σε περίπτωση εκτέλεσης πολλαπλών γράφων, και στην περίπτωση οπτικοποιήσης της διαδικασίας εκπαίδευσης (βλ. Tensorboard).

Υπολογισμός μέσων όρων (averaging)

Ο υπολογισμός των μέσων όρων των παραμέτρων από τα αποτελέσματα των διαφορετικών workers, είναι η απλούστερη εκδοχή του παραλληλισμού δεδομένων. Ο αλγόριθμος δουλεύει ως εξής:

1. Τυχαία αρχικοποίηση παραμέτρων, με βάση την μορφή του μοντέλου
2. Διανομή ενός αντιγράφου των παραμέτρων σε κάθε worker
3. Εκπαίδευση κάθε worker σε ένα κομμάτι (batch) δεδομένων
4. Ενημέρωση των κεντρικών παραμέτρων με τους μέσους όρους των παραμέτρων που κρατούνται στους workers
5. Όσο υπάρχουν περισσότερα δεδομένα για επεξεργασία, πήγαινε στο βήμα 2.

Τα βήματα 2 έως 4 παρουσιάζονται στην παρακάτω εικόνα. Στο διάγραμμα αυτό, το W αντιπροσωπεύει τις παραμέτρους (weights, biases) στο δίκτυο. Τηρείται χρονολογική σειρά παραλλαγής των παραμέτρων, όπου χρειάζεται, για κάθε worker.



Εικόνα 16: Μέσος όρος - Ασύγχρονος παραλληλισμός δεδομένων

ΚΕΦΑΛΑΙΟ 3

TENSORFLOW ON MOBILE PLATFORMS

Το Tensorflow σχεδιάστηκε για να είναι μια καλή λύση deep learning για κινητές πλατφόρμες. Επί του παρόντος, υπάρχουν δύο εκδοχές για την ανάπτυξη εφαρμογών μηχανικής μάθησης σε κινητές και ενσωματωμένες συσκευές: Tensorflow for mobiles και Tensorflow Lite.

Παραδοσιακά το deep learning έχει συνδεθεί με datacenters και γιγάντια clusters από υψηλής ισχύος μηχανές GPU. Ωστόσο η χρονοβόρα αποστολή όλων των δεδομένων από μια συσκευή που έχει πρόσβαση μέσω δικτύου στα clusters αυτά μπορεί να είναι πολύ δαπανηρή. Η εκτέλεση σε κινητά καθιστά δυνατή την παροχή πολύ διαδραστικών εφαρμογών με τρόπο που δεν είναι δυνατό όταν συμπεριλαμβάνεται ένα ταξίδι μετ'επιστροφής μέσω δικτύου. Μερικές συνήθεις περιπτώσεις χρήσης μηχανικής μάθησης σε κινητές συσκευές:

Αναγνώριση φωνής: Υπάρχουν πολλές ενδιαφέρουσες εφαρμογές που μπορούν να κατασκευαστούν με μια διασύνδεση που βασίζεται στην ομιλία, και πολλές από αυτές απαιτούν επεξεργασία **στη συσκευή**. Τις περισσότερες φορές ένας χρήστης δεν δίδει συγκεκριμένες εντολές, και η συνεχής ροή ήχου προς έναν απομακρυσμένο διακομιστή θα ήταν σπατάλη εύρους ζώνης, καθώς θα περιείχε ως επί το πλείστον σιωπή ή και θορύβους. Για την επίλυση αυτού του προβλήματος συνηθίζουμε να έχουμε ένα μικρό νευρωνικό δίκτυο που εκτελεί on-device ακρόαση για μια συγκεκριμένη λέξη-κλειδί. Αφού εντοπιστεί η λέξη-κλειδί, η υπόλοιπη συνομιλία μπορεί να μεταδοθεί στον διακομιστή για περαιτέρω επεξεργασία, εάν απαιτείται περισσότερη υπολογιστική ισχύς. Χαρακτηριστικό παράδειγμα αποτελούν τα λεκτικά κλειδιά των ψηφιακών βοηθών όπως για παράδειγμα «Ok, Google» για το Google Assistant, «Hey Siri» για την Siri του iOS και «Hey Cortana» για την Cortana της Mircosoft.

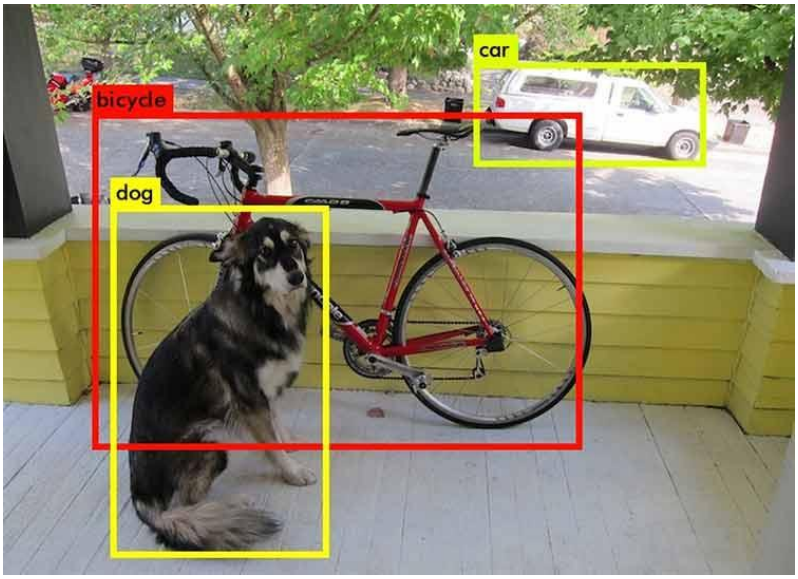
Αναγνώριση εικόνας: Μπορεί να είναι πολύ χρήσιμο για μια εφαρμογή σε κινητά να μπορεί να βγάζει νόημα για μια εικόνα που «βλέπει» η κάμερα. Εάν οι χρήστες τραβούν φωτογραφίες, η αναγνώριση του περιεχομένου τους μπορεί να βοηθήσει τις εφαρμογές φωτογραφικής μηχανής να εφαρμόσουν τα κατάλληλα φίλτρα ή να επισημάνουν τις φωτογραφίες έτσι ώστε να είναι εύκολα κατανοητές. Είναι σημαντικό και για ενσωματωμένες εφαρμογές, αφού μπορούν να χρησιμοποιηθούν δίκτυα-αισθητήρες εικόνας για να ανιχνευθεί κάθε είδους ενδιαφέρουσα συνθήκη. Για παράδειγμα να εντοπίζει απειλούμενα ζώα σε άγρια κατάσταση ή να αναφέρει πόσο αργά τρέχει ένα όχημα.

Το Tensorflow έρχεται με πολλά παραδείγματα αναγνώρισης διαφόρων τύπων αντικειμένων μέσα σε εικόνες μαζί με μια ποικιλία διαφορετικών προ-εκπαιδευμένων μοντέλων και όλα αυτά μπορούν να **λειτουργούν σε κινητές συσκευές**.

Εντοπισμός αντικειμένων: Μερικές φορές είναι σημαντικό να γνωρίζουμε πού βρίσκονται τα αντικείμενα σε μια εικόνα καθώς και ποιά είναι αυτά. Υπάρχουν πολλές περιπτώσεις πραγματικής χρήσης που θα μπορούσαν να ωφελήσουν μια εφαρμογή για κινητά, όπως η καθοδήγηση των χρηστών στο σωστό στοιχείο, όταν τους προσφέρεται βοήθεια για τον καθορισμό του ασύρματου δικτύου τους ή για την παροχή ενημερωτικών ετικετών επάνω στις λειτουργίες τοπίου. Οι

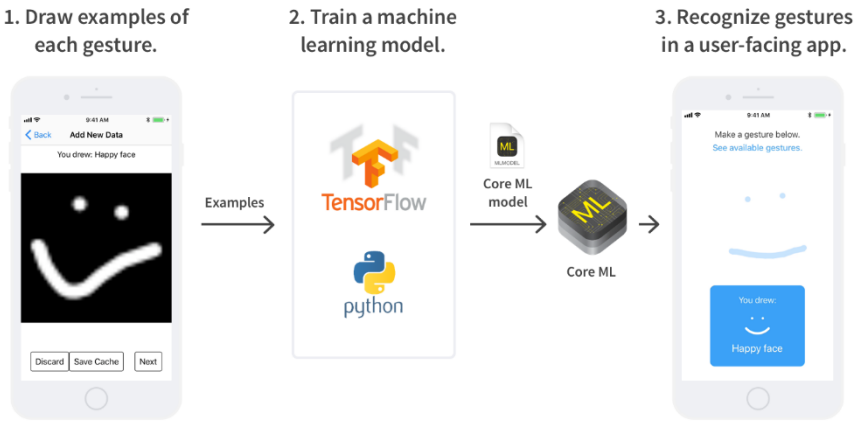
ενσωματωμένες εφαρμογές πρέπει συχνά να υπολογίζουν τα αντικείμενα που περνούν από κοντά, είτε πρόκειται για παράσιτα σε έναν τομέα καλλιεργειών είτε για ανθρώπους, αυτοκίνητα και ποδήλατα που περνούν πέρα από έναν λαμπτήρα δρόμου.

Το Tensorflow προσφέρει ένα προρυθμισμένο μοντέλο για το σχεδιασμό πλαισίων οριοθέτησης γύρω από άτομα που ανιχνεύονται σε εικόνες, μαζί με τον κώδικα παρακολούθησης για να τα ακολουθούμε με την πάροδο του χρόνου. Η παρακολούθηση είναι ιδιαίτερα σημαντική για εφαρμογές όπου προσπαθούμε να μετρήσουμε πόσα αντικείμενα παρουσιάζονται με την πάροδο του χρόνου, καθώς δίνει μια ειδοποίηση (callback) όταν ένα νέο αντικείμενο εισέρχεται ή εξέρχεται από τη σκηνή.



Εικόνα 17: Tensorflow mobile: αναγνώριση αντικειμένων

Αναγνώριση χειρονομίας: Σε μερικές περιπτώσεις είναι χρήσιμο να ελέγχουμε εφαρμογές με χειρονομίες, οι οποίες αναγνωρίζονται είτε από εικόνες είτε αναλύοντας δεδομένα αισθητήρα επιταχυνσιόμετρου (accelerometer). Το Tensorflow είναι ένας αποτελεσματικός τρόπος για την ανάπτυξή τέτοιων μοντέλων και εφαρμογών, καθώς και χρήση τους σε κινητές συσκευές.



Εικόνα 18: Tensorflow mobile - αναγνώριση χειρονομίας

Μετάφραση: Η μετάφραση από τη μια γλώσσα στην άλλη γρήγορα και με ακρίβεια, ακόμα και αν δεν είναι εφικτή σύνδεση στο δίκτυο, αποτελεί μια σημαντική περίπτωση χρήσης. Τα βαθιά δίκτυα

είναι πολύ αποτελεσματικά σε αυτό το είδος εργασιών. Συχνά χρησιμοποιούνται επαναλαμβανόμενα μοντέλα ακολουθία-προς-ακολουθία, όπου μπορεί να εκτελεστεί ένας μόνο γράφος για να κάνει ολόκληρη τη μετάφραση, χωρίς να χρειάζεται να εκτελεστούν ξεχωριστά στάδια ανάλυσης. Μια ευρέως γνωστή εφαρμογή μετάφρασης, αποτελεί το Google Translate σε κινητά, η οποία όπως οι περισσότερες εφαρμογές της Google χρησιμοποιεί τα προαναφερθέντα μοντέλα.

Ταξινόμηση κειμένου: Στις περιπτώσεις που χρειάζεται να κάνουμε σχετικές προτροπές στους χρήστες με βάση αυτό που πληκτρολογούν ή διαβάζουν, είναι πολύ χρήσιμο να μπορεί η εφαρμογή να κατανοήσει την έννοια του κειμένου. Σε αυτό το σημείο εμφανίζεται η ταξινόμηση κειμένου. Η ταξινόμηση κειμένου είναι ένας όρος ομπρέλα που καλύπτει τα πάντα, από την ανάλυση του συναισθήματος έως την ανακάλυψη θεματικών.

Μηχανική μάθηση σε κινητές πλατφόρμες & νέφος

Αυτά τα παραδείγματα εφαρμογών δίνουν μια ιδέα για το πως τα βαθιά δίκτυα σε κινητές συσκευές συμπληρώνουν τις υπηρεσίες νέφους (cloud). Μπορεί στην υπολογιστική νέφους να χρησιμοποιείται η υπολογιστική δύναμη σε ένα απόλυτα ελεγχόμενο και επεκτάσιμο περιβάλλον, όμως η μηχανική μάθηση σε κινητές συσκευές προσφέρει μεγαλύτερη διαδραστικότητα. Σε περιπτώσεις όπου το νέφος δεν είναι προσβάσιμο, ή η χωρητικότητά του είναι περιορισμένη, μπορούμε να προσφέρουμε μια εμπειρία εκτος σύνδεσης, ή να μειώσουμε τον φόρτο δικτύου και νέφους, εκτελώντας απλές λειτουργίες τοπικά στην φορητή συσκευή.

Η εκτέλεση υπολογισμών στην συσκευή, μπορεί επίσης να επιδείξει πότε χρειάζεται η μετάβαση στο νέφος. Ένα καλό παράδειγμα είναι η αναγνώριση «καυτών» λέξεων-κλειδιών σε ομιλία. Από την στιγμή που οι συσκευές μπορούν να ακούν συνεχόμενα ομιλία μέχρι να ξεχωρίσουν μια λέξη κλειδί, και στη συνέχεια να μεταφέρουν την υπόλοιπη ροή ήχου στο νέφος για αναγνώριση φωνής. Χωρίς το κομμάτι αναγνώρισης στην συσκευή, η λειτουργία όλης της εφαρμογής δεν θα ήταν εφικτή, και αυτό το μοτίβο επαναλαμβάνεται σε πολλές περιπτώσεις χρήσης μηχανικής μάθησης σε κινητές συσκευές. Το να αναγνωριστεί οτι η είσοδος σε έναν αισθητήρα είναι αρκετά ενδιαφέρουσα για να επεξεργαστεί περαιτέρω, δημιουργεί πολλές χρήσιμα πιθανά προϊόντα.

Πλατφόρμες για το Tensorflow

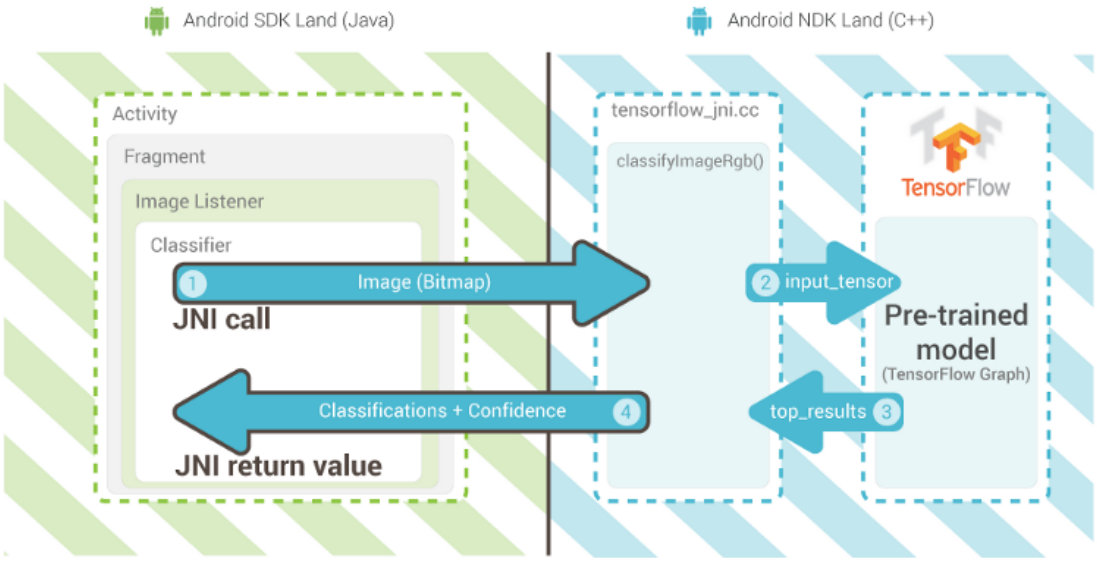
Το Tensorflow λειτουργεί σε Ubuntu Linux, Windows 10, και OS X. Σε αυτά τα λειτουργικά μπορεί να χρησιμοποιηθεί για να εκπαιδεύσει και να δοκιμάσει δίκτυα, είτε για να εκτελέσει γράφους για προβλέψεις και εξαγωγή δεδομένων. Τα εκπαιδευμένα μοντέλα που δημιουργεί το Tensorflow, μπορούν έπειτα να ενσωματωθούν σε κινητές εφαρμογές και να χρησιμοποιηθούν για προβλέψεις και εξαγωγή δεδομένων σε **κινητές** πλέον **συσκευές.** Με κατάλληλη επεξεργασία στο περιβάλλον της Python, όπου εκπαιδεύεται το δίκτυο, υπάρχει η δυνατότητα να «παγώνσει» (freeze) το μοντέλο και να αποθηκευτεί σε ένα αρχείο με κατάληξη [*.pb] . Αυτό το αρχείο περιέχει ένα αντίγραφο του γράφου έτοιμο για εκτέλεση, είτε από το πλήρες Tensorflow στον ίδιο ή σε άλλο υπολογιστή / server είτε από κινητές εφαρμογές **Android** & **iOS**. Να σημειωθεί οτι είναι εφικτό για λόγους μεγέθους και χωρητικότητας να αφαιρέσουμε από το αρχείο, κώδικα και κομμάτια του γράφου που χρειάζονται μόνο στην εκπαίδευση, εφόσον στην κινητή πλατφόρμα ο γράφος θα χρησιμοποιηθεί μόνο για αναγνώριση, και όχι για περαιτέρω εκπαίδευση.

3.1 Android

Κατά την αρχή αναζήτησης πληροφοριών και εκπόνησης της παρούσας εργασίας,ο μοναδικός τρόπος να χρησιμοποιηθεί ένα προ-εκπαιδευμένο σε desktop περιβάλλον μοντέλο, μέσα σε μια android εφαρμογή, ήταν το **Tensorflow Mobile**. Από την αρχή του 2018 ανακοινώθηκε και αργότερα παρουσιάστηκε μια πιο απλουστευμένη για τον προγραμματιστή εφαρμογών μορφή του Tensorflow, η οποία μάλιστα είναι ικανή να «τρέξει» και σε ενσωματωμένα συστήματα, εκτός από Andorid και iOS συσκευές. Ονομάζεται **Tensorflow Lite** και αποτελεί την εξέλιξη της mobile έκδοσης.

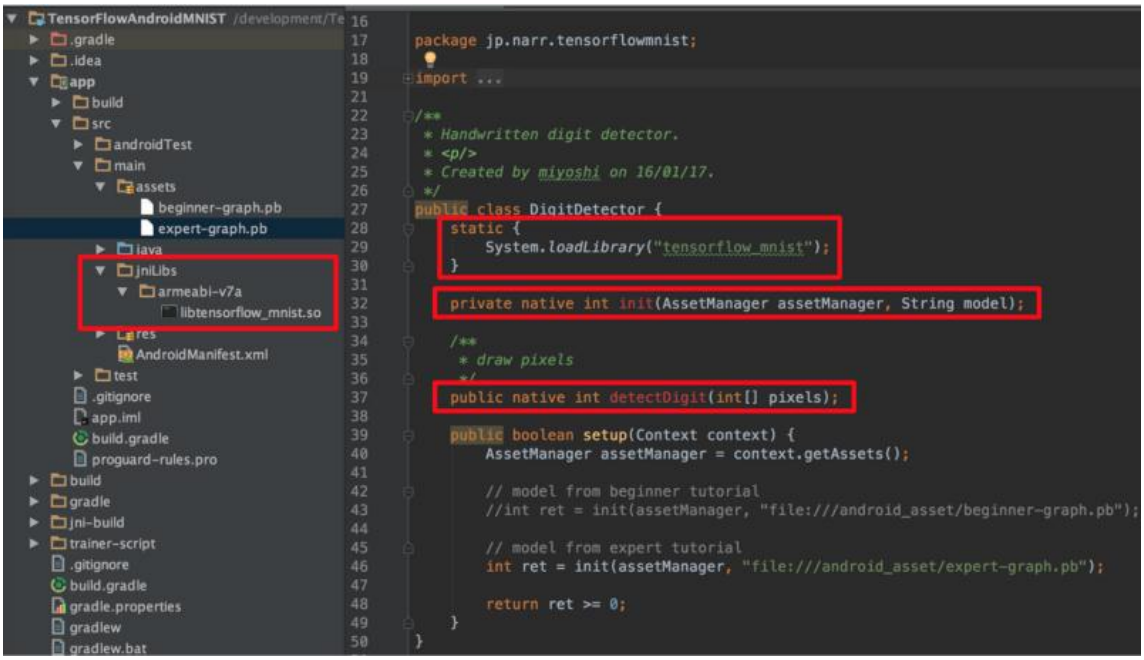
3.1.1 Tensorflow Mobile

Από τις πρώτες εκδόσεις του Tensorflow mobile για android, η λογική ήθελε να χρησιμοποιήσει το εκπαιδευμένο μοντέλο για αναγνώριση προτύπων μέσα σε μια εφαρμογή. Η βασική δυσκολία έγκειτο στο γεγονός οτι η συνηθέστερη επιλογή γλώσσας ενός προγραμματιστή android εφαρμογών ήταν η Java (αργότερα έκανε την εμφάνισή της η Kotlin – που γίνεται ολοένα πιο δημοφιλής) ενώ το Tensorflow mobile είναι γραμμένο σε C++. Για να λυθεί αυτό το πρόβλημα η Google διέθετε το C++ Tensorflow, παράλληλα με την χρήση διεπαφών σε Java, ώστε να «περάσουν» τα δεδομένα από μια κλήση Java συνάρτησης στο Tensorflow, μέσω της Java Native Interface (JNI) και του Android Nateve Development Kit (NDK).



Εικόνα 19: Tensorflow mobile using JNI

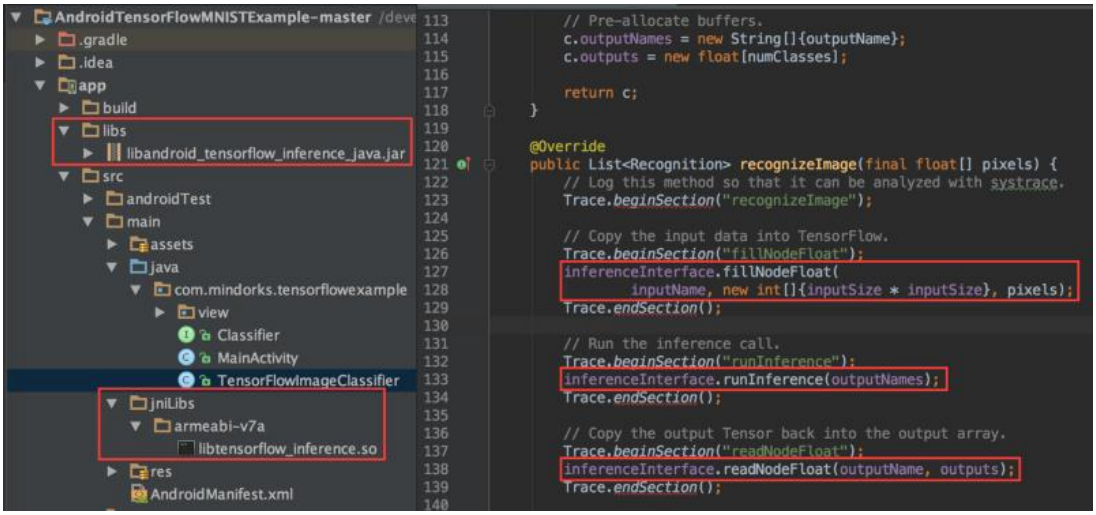
Για παράδειγμα – με το MNIST dataset, μια Java κλάση, όπως η DigitDetector της εικόνας 24, χρησιμοποιεί την native library **libTensorflow_mnist.so** για να διαβάσει και να τρέξει τον γράφο που βρίσκεται στο *.pb αρχείο μαζί με το αρχείο *.txt που περιέχει τα αντίστοιχα labels, μέσα στα android assets, τα τοπικά αρχεία δηλαδή που έχει πρόσβαση και χρησιμοποιεί η εφαρμογή.



Εικόνα 20: Χρήση pre-compiled βιβλιοθήκης JNI

Αυτός ο τρόπος ανάπτυξης εφαρμογών Tensorflow είναι αρκετά πολύπλοκος και αρκετά δύσκολος στην συντήρησή του. Για παράδειγμα ο JNI κώδικας χρειάζεται να συσταθεί (build) ξεχωριστά από τον Java κώδικα, δηλαδή με έξτρα εργαλεία (build tools) – όπως το bazel, πέρα από τα Java εργαλεία του Android Studio (εξ ορισμού το gradle). Επιπλέον, αυτός ο τρόπος είναι πολύ περιοριστικός ως προς την αρχιτεκτονική του επεξεργαστή στον οποίο θα εκτελεστεί η εφαρμογή, αφού η διεπαφή δημιουργήθηκε σε αυτό το παράδειγμα για επεξεργαστές με αρχιτεκτονική (instruction set architecture) **arm-v7.** Πρακτικά αυτό σημαίνει οτι για να στοχεύσει η εφαρμογή περισσότερες συσκευές με διαφορετικές αρχιτεκτονικές επεξεργαστών (όπως x86, armv8 κτλ) θα πρέπει να συμπεριλάβει εκ νέου τις αντίστοιχες διεπαφές.

Για τους παραπάνω λόγους, η Google αποφάσισε να ορίσει λίγο καλύτερα τον τρόπο με τον οποίο θα μιλάνε οι Java εφαρμογές με το Tensorflow. Δημιούργησε μια Java class, την `TensorflowInferenceInterface`, και την συνόδευσε με έτοιμες, προ-συσταθείσες βιβλιοθήκες `libandroid_Tensorflow_inference_java.jar` (την διεπαφή σε Java επίπεδο) και `libTensorflow_inference.so` (τον JNI κώδικα που μιλάει με το Tensorflow μοντέλο). Έτσι οργάνωσε τον κώδικα όπως στην εικόνα 25, δίνοντας στον android developer την δυνατότητα να ασχοληθεί ακόμα λιγότερο με Tensorflow και low-level κώδικα, αλλά να «περάσει» δεδομένα σε μια java κλάση και να έχει από την ίδια τα αποτελέσματα άμεσα.



Εικόνα 21: Tensorflow Inferecne Interface

Αργότερα, όλα τα παραπάνω συμπτύχθηκαν σε μια βιβλιοθήκη *.aar, η οποία μπορεί να χρησιμοποιηθεί απευθείας σαν εξάρτηση (dependency) μέσω του gradle – του προκαθορισμένου build tool του Android Studio. Με αυτό τον τρόπο, μια απλή γραμμή

```
compile 'org.Tensorflow:Tensorflow-android:+'
```

μέσα στο build.gradle αρχείο αρκεί για να μπορεί ο προγραμματιστής να χρησιμοποιήσει την Java class `TensorflowInferenceInterface` και το μοντέλο του.

3.1.2 Tensorflow Lite

Το Tensorflow Lite αποτελεί την εξελιγμένη και ακόμα πιο «ελαφριά» (lightweight) έκδοση του Tensorflow mobile, η οποία μπορεί να χρησιμοποιηθεί και σε ενσωματωμένα συστήματα. Προσφέρει αποτελέσματα μηχανικής μάθησης στις φορητές συσκευές, με πολύ μικρή καθυστέρηση απόκρισης και ελάχιστο μέγεθος (low binary size). Ακόμα, μπορεί να εκμεταλλευτεί το υλικό της συσκευής όταν αυτό προσφέρεται και να επιταχύνει διαδικασίες, μέσω του Android Neural Networks API. Το Tensorflow Lite επίσης, χρησιμοποιεί πολλές τεχνικές για την επίτευξη χαμηλής καθυστέρησης, όπως βελτιστοποίηση των πυρήνων (kernels) για κινητές εφαρμογές, προ-συγχωνευμένες ενεργοποιήσεις και κβαντισμένους πυρήνες που επιτρέπουν μικρότερα και ταχύτερα μαθηματικά μοντέλα σταθερής υποδιαστολής.

Neural Networks API (NNAPI)

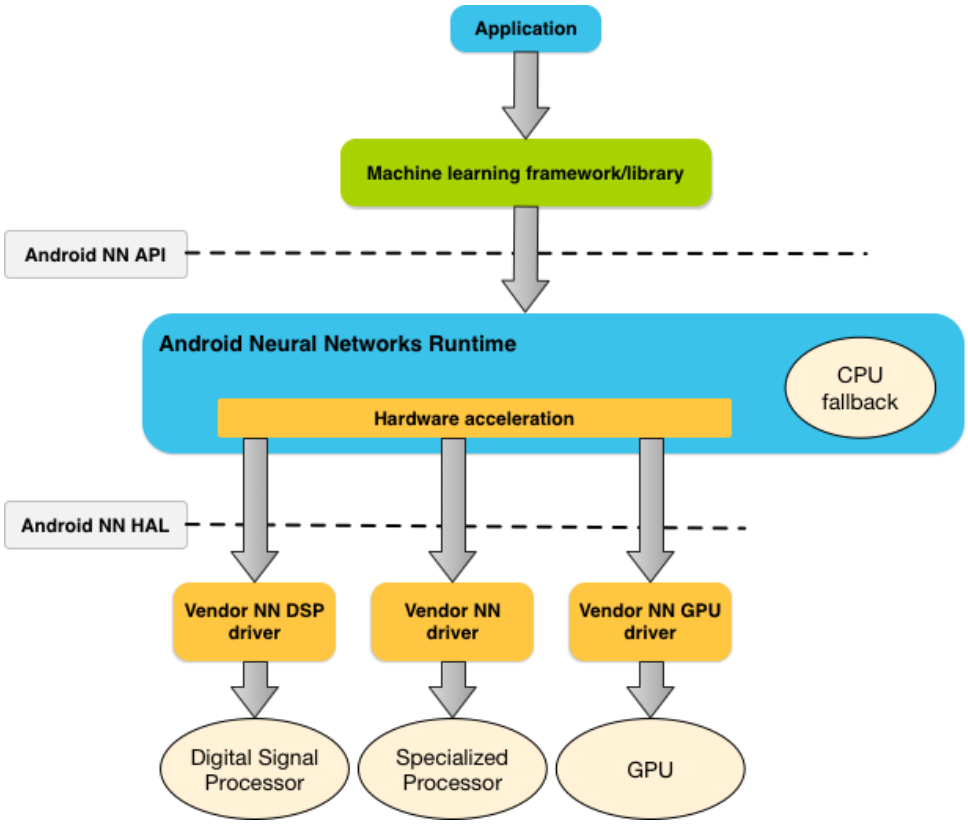
Πρόκειται για ένα API του Android σε C που έχει σχεδιαστεί για την εκτέλεση υπολογιστικών λειτουργιών για μηχανική μάθηση σε κινητές συσκευές. Το NNAPI παρέχει ένα βασικό στρώμα λειτουργικότητας για πλαίσια ανώτερης μηχανικής μάθησης (όπως Tensorflow Lite και άλλα) που κατασκευάζουν και εκπαιδεύουν νευρωνικά δίκτυα. Το API είναι διαθέσιμο σε όλες τις συσκευές με Android 8.1 (επίπεδο 27 API) ή υψηλότερη. Υποστηρίζει τη χρήση «συμβολισμών» εφαρμώζοντας δεδομένα από συσκευές Android σε προηγουμένως εκπαιδευμένα μοντέλα που ορίζονται από προγραμματιστές. Συνήθεις περιπτώσεις αντιστοίχισης (συμβολισμών)

περιλαμβάνουν την ταξινόμηση εικόνων, την πρόβλεψη συμπεριφοράς χρηστών και την επιλογή κατάλληλων απαντήσεων σε ένα ερώτημα αναζήτησης.

Το NNAPI είναι κατασκευασμένο για να καλείται από βιβλιοθήκες μηχανών μάθησης, πλαίσια και εργαλεία που επιτρέπουν στους προγραμματιστές να εκπαιδεύσουν τα μοντέλα τους εκτός συσκευής και να τα χρησιμοποιήσουν σε συσκευές Android. Οι εφαρμογές συνήθως δεν χρησιμοποιούν απευθείας το NNAPI, αλλά απευθύνονται σε αυτό μέσω πλαισίων ανώτερης μηχανικής μάθησης. Αυτά τα πλαίσια, με τη σειρά τους, θα μπορούσαν να χρησιμοποιήσουν το NNAPI για να πραγματοποιήσουν λειτουργίες συμπερασμάτων με επιτάχυνση υλικού στις υποστηριζόμενες συσκευές (Android Oreo images or higher).

Με βάση τις απαιτήσεις της εφαρμογής και τις δυνατότητες υλικού σε μια συσκευή, μπορεί να διανέμει αποτελεσματικά το φόρτο εργασίας υπολογισμών μηχανικής μάθησης σε όλους τους επεξεργαστές που βρίσκονται στη διάθεσή του, συμπεριλαμβανομένων των εξειδικευμένων υλικών του νευρικού δικτύου, των μονάδων επεξεργασίας γραφικών (GPU) και των ψηφιακών επεξεργαστών σημάτων (DSPs) .

Για συσκευές που δεν διαθέτουν εξειδικευμένο πρόγραμμα οδήγησης προμηθευτή, το πρόγραμμα χρόνου εκτέλεσης NNAPI βασίζεται σε βελτιστοποιημένο κώδικα για την εκτέλεση αιτημάτων από την διαθέσιμη CPU.



Εικόνα 22: Δομή NNAPI

Χρήση Tensorflow lite

Για να χρησιμοποιηθεί το Tensorflow Lite σε μια android εφαρμογή, μπορεί είτε να συμπεριληφθεί με την προαναφερθείσα εξάρτηση (dependency) **compile 'org.Tensorflow:Tensorflow-android:+'** είτε να συσταθεί από τον πηγαίο κώδικα (build from source) με την χρήση του κατάλληλου εργαλείου (build tool – bazel). Και στις δύο περιπτώσεις απαιτείται να υπάρχει εγκατεστημένη η τελευταία έκδοση Android Software Development Kit [28] (SDK) καθώς και η τελευταία Android Native Development Kit [14] (NDK) εφόσον η διεπαφή του Tensorflow είναι γραμμένη ,όπως αναφέραμε, σε C++.

Για να χρησιμοποιηθεί το Tensorflow μοντέλο στην εφαρμογή αρκούν μερικές γραμμές:

```
static Classifier create(AssetManager assetManager,
                         String modelPath,
                         String labelPath,
                         int inputSize) throws IOException {
   TensorflowImageClassifier classifier = new TensorflowImageClassifier();
   classifier.interpreter = new
Interpreter(classifier.loadModelFile(assetManager, modelPath));
   classifier.labelList = classifier.loadLabelList(assetManager, labelPath);
   classifier.inputSize = inputSize;
   return classifier;
}

/*********/

classifier = TensorflowImageClassifier.create(getAssets(),MODEL_PATH, LABEL_PATH,
INPUT_SIZE);
final List<Classifier.Recognition> results = classifier.recognizeImage(camera.getBitmap());
```

Tensorflow Graphs file formats

Τα στάδια εκπαίδευσης του Tensorflow παράγουν αντικείμενα τύπου `tf.GraphDef`, που αντιπροσωπεύουν τον Tensorflow γράφο και περιέχουν operators, tensors και μεταβλητές. Αποθηκεύονται σε αρχεία *.pb ή *.pbtxt – Google protobufs (δομή αναπαράστασης δεδομένων ανεξαρτήτου πλαρφόρμας και γλώσσας προγραμματισμού). Κατα την διάρκεια της εκπαίδευσης, το Tensorflow χρησιμοποιεί τα αρχεία checkpoints, για να κρατήσει σειριοποιημένες τις τιμές των μεταβλητών στα στάδια εκπαίδευσης. Πρόκειται για αρχεία *.ckpt τα οποία περιέχουν μόνο μεταβλητές, γεγονός που σημαίνει οτι δεν μπορούν να αναπαραστήσουν γράφο από μόνα τους. Επίσης, τα FrozenGraphDef αντικείμενα, αποτελούν υποκατηγορία του GraphDef που δεν περιέχει μεταβλητές. Ένα GraphDef μπορεί να μετατραπεί σε FrozenGraphDef λαμβάνοντας ένα CheckPoint και ένα GraphDef και μετατρέποντας κάθε μεταβλητή σε μια σταθερά χρησιμοποιώντας την τιμή που ανακτάται από το CheckPoint. Τέλος, η μορφή SavedModel, περιέχει ένα GraphDef και ένα Checkpoint, αντιστοιχίζοντας input & output arguments σε ένα μοντέλο.

Μια βασική διαφοροποίηση της lite έκδοσης του Tensorflow από την πρόγονό της (Tensorflow mobile) είναι η μορφή του μοντέλου πουχρησιμοποιεί. Συγκεκριμένα λειτουργεί με μοντέλα στην μορφή *.tflite, που αποτελούν μια ελαφρώς διαφοροποιημένη εκδοχή του FrozenGraphDef,

μοντέλο δηλαδή που δεν περιέχει τιμές μεταβλητών (checkpoints), και είναι βελτιστοποιημένο ταχύτητα και μικρό μέγεθος (binary size). Η βασική διαφοροποίηση έγκυται στο πρωτόκολλο σειριοποίησης, όπου χρησιμοποιείται το **FlatBuffer** αντί για **Google ProtoBuf.** Το κύριο πλεονέκτημα των FlatBuffers προέρχεται από το γεγονός ότι μπορούν να χαρτογραφηθούν στην μνήμη και να χρησιμοποιηθούν απευθείας από το δίσκο χωρίς να φορτωθούν και να αναλυθούν. Αυτό δίνει πολύ γρηγορότερους χρόνους εκκίνησης και δίνει στο λειτουργικό σύστημα τη δυνατότητα φόρτωσης και διαγραφής των απαιτούμενων σελίδων από το αρχείο μοντέλου, αντί να τερματίζει την εφαρμογή όταν ξεπεραστούν τα μέγιστα όρια χρήσης μνήμης. Πρακτικά, αποτελεί μέθοδο καλύτερης διαχείρισης των πόρων του συστήματος για φορητές συσκευές.

TOCO: Tensorflow Lite Optimizing Converter

Οι κινητές συσκευές έχουν σημαντικούς περιορισμούς – κυρίως λόγω μεγέθους μνήμης και επεξεργαστικής ισχύος, επομένως αξίζει να εξεταστεί κάθε προεπεξεργασία που μπορεί να γίνει για να μειωθεί το αποτύπωμα μιας εφαρμογής. Με το Tensorflow lite ενσωματώνεται ένας νέος μετατροπέας γράφων κατά την εγκατάσταση. Πρόκειται για ένα βοηθητικό πρόγραμμα που ονομάζεται "Tensorflow Lite Optimizing Converter" – **TOCO**. Το toco εφαρμόζει αρκετές βελτιστοποιήσεις που είναι χρήσιμες για την μετατροπή από Tensorflow γράφο σε γράφο για Tensorflow lite (.tflite). Οι μετατροπές αφορούν τις περικοπές μερών του γράφου που δεν κρίνονται απαραίτητα για την κινητή συσκευή, και την βέλτιστη «συγχώνευση» και τροποποίηση των λειτουργιών (operations) ώστε να εκτελείται ταχύτερα και με μικρότερο αντίκτυπο στους πόρους της συσκευής. Οι περικοπές είναι ιδιαίτερα απαραίτητες, δεδομένου του οτι η lite έκδοση του Tensorflow δεν έχει δυνατότητες εκπαίδευσης, συνεπώς οι εργασίες εκαπίδευσης είναι περιττές στον lite γράφο. Παρ’ότι το TOCO θα μπορούσε να χρησιμοποιηθεί για τη βελτιστοποίηση των κανονικών αρχείων graph.pb, το TFLite χρησιμοποιεί διαφορετική μορφή serialization από το κανονικό Tensorflow (FlatBuffers). Συνεπώς το TOCO εξάγει απευθείας αρχεία *.tflite για χρήση αποκλειστικά σε Tensorflow lite βιβλιοθήκη.

3.2 iOS

Μέχρι το 2017 ο μόνος τρόπος για να χρησιμοποιήσεις ένα μοντέλο μηχανικής μάθησης μέσα σε μία iOS εφαρμογή ήταν να χρησιμοποιήσεις βιβλιοθήκες σαν την Accelerate και Metal, οι οποίες είναι βιβλιοθήκες, που ειδικεύονται στους πολλαπλούς μαθηματικούς υπολογισμούς μεγάλης έκτασης, μέσω της βέλτιστης αξιοποιήσεις του hardware ενός υπολογιστή(GPU, CPU). Με αυτό τον τρόπο θα έπρεπε να γραφτεί από το μηδέν ένα μοντέλο μηχανικής μάθησης όπως και ο τρόπος που θα αξιοποιηθεί μέσα στην εφαρμογή. Στο WWDC17 το μεγαλύτερο ετήσιο συνέδριο της Apple, ανακοινώθηκε η βιβλιοθήκη CoreML η οποία θα υποστηρίζεται από τα παρακάτω προϊόντα της Apple και της αντίστοιχες εκδόσεις τους:

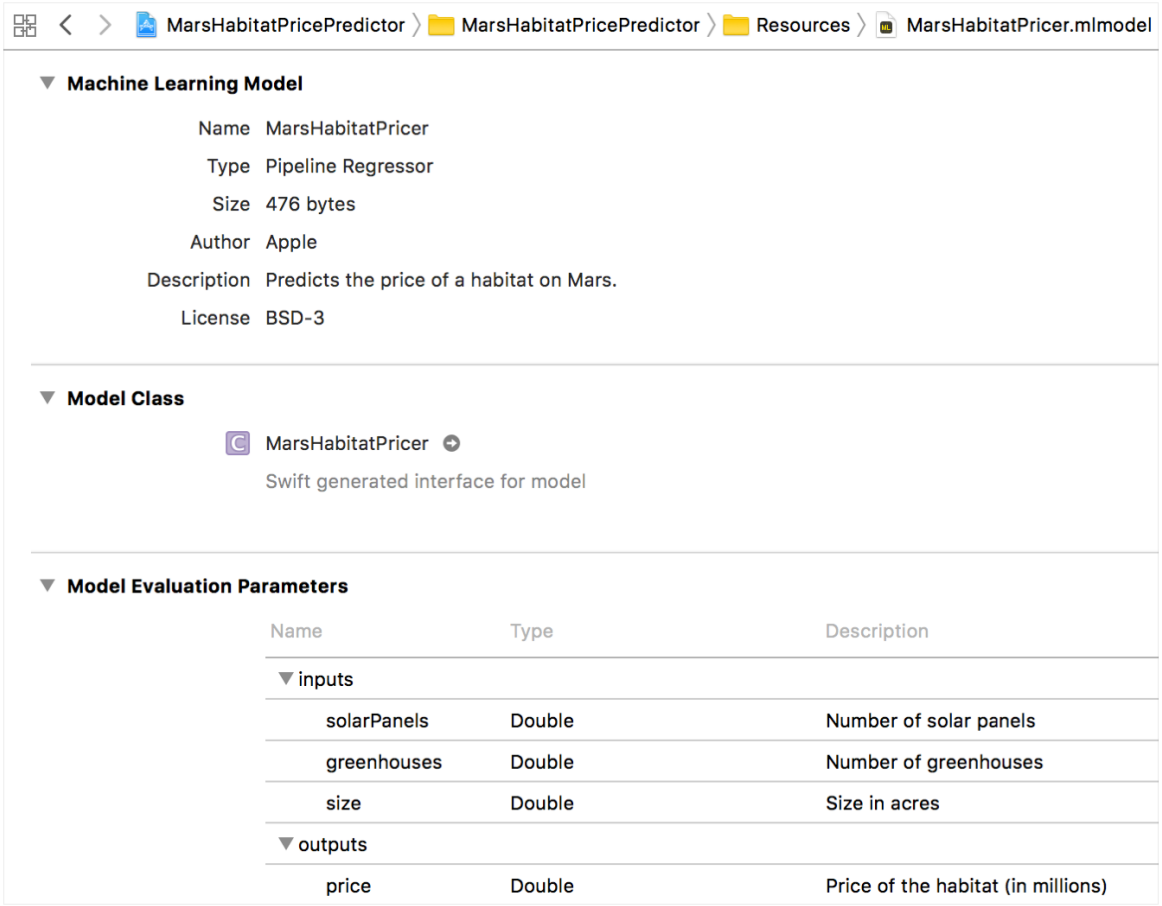
- iOS 11.0+
- macOS 10.13+
- tvOS 11.0+
- watchOS 4.0+

Το CoreML απλουστεύει την διαδικασία ενσωμάτωσης ενός μοντέλου μηχανικής μάθησης μέσα σε μία εφαρμογή και χτίστηκε πάνω στης βιβλιοθήκες Accelerate και Metal έτσι ώστε να δοθεί έμφαση στην οικονομία των πόρων μίας συσκευής εφόσον αυτές οι βιβλιοθήκες χρησιμοποιούν API της C ειδικά γραμμένα για βελτιστοποίηση αλγορίθμων και μαθηματικών υπολογισμών.

3.2.1 CoreML

Το CoreML δίνει την ικανότητα να πάρει κάποιος ένα ήδη εκπαιδευμένο μοντέλο από ένα μεγάλο εύρος από διάφορες πλατφόρμες και να το εφαρμόσει μέσα στην εφαρμογή του. Δέχεται αυτά τα μοντέλα σε μορφή αρχείων τύπου .mlmodel για τα οποία το CoreML παράγει wrapper κλάσεις κώδικα είτε σε Swift, είτε σε Objective-C. Αυτές οι κλάσσεις δίνουν πρόσβαση σε κάποιες παραμέτρους του μοντέλου και δίνουν πίσω κάποιο αποτέλεσμα. Τα μοντέλα αυτά μπορούν να παραμετροποιηθούν είτε μέσω του επίσημου IDE της Apple το Xcode, είτε μέσω κώδικα.

Ενα πάραδειγμα εισαγωγής ενός προ-εκπαιδευμένου γραμμικού μοντέλου πρόβλεψης κόστους και χρήσης του:



Εικόνα 23: Imported mlmodel in Xcode

```
let model = MarsHabitatPricer()

guard let marsHabitatPricerOutput = try? model.prediction(solarPanels:
solarPanels, greenhouses: greenhouses, size: size) else {
    fatalError("Unexpected runtime error.")
}

let price = marsHabitatPricerOutput.price
priceLabel.text = priceFormatter.string(for: price)
```

Άμα το μοντέλο έχει δημιουργηθεί μέσω μίας εξωτερικής πλατφόρμας/βιβλιοθήκης μηχανικής μάθησης τότε το CoreML παρέχει το εργαλείο coremltools το οποίο δίνει την δυνατότητα μετατροπής κάποιον τέτοιον μοντέλων σε .mlmodel αρχεία τα οποία μπορούν να χρησιμοποιηθούν από το Xcode.

Κάποιες από αυτές της πλατφόρμες, τα εργαλεία και τα ήδη μοντέλων καταγράφονται παρακάτω:

Model type	**Supported models**	**Supported frameworks**
Neural networks	Feedforward, convolutional, recurrent	Caffe v1 Keras 1.2.2+
Tree ensembles	Random forests, boosted trees, decision trees	scikit-learn 0.18 XGBoost 0.6
Support vector machines	Scalar regression, multiclass classification	scikit-learn 0.18 LIBSVM 3.22
Generalized linear models	Linear regression, logistic regression	scikit-learn 0.18
Feature engineering	Sparse vectorization, dense vectorization, categorical processing	scikit-learn 0.18
Pipeline models	Sequentially chained models	scikit-learn 0.18

3.2.2 General Purpose Machine Learning Frameworks and repositories for iOS and MacOS

Με την διάδοση του Machine Learning σαν τεχνολογία φυσικό επόμενο είναι το open source community του iOS και του macOS να δημιουργήσει μια μεγάλη γκάμα από open source projects η οποία πληθαίνει καθημερινά. Κάποια από τα σημαντικότερα από αυτά τα project αναφέρονται παρακάτω:

3.2.2.1 Objective-C

- **YMCL** - Το YMCL είναι μια βιβλιοθήκη γραμμένη σε Objective-c η οποία μπορεί να χρησιμοποιήθει και απο την Swift. Παρέχει εννέα εφαρμογές αλγορίθμων μηχανικής μάθησης για Supervised Learning, δύο γενετικούς αλγορίθμους βελτιστοποίησης πολλαπλών στόχων (**Computational (Multi-Objective) Optimization**), ένα αλγόριθμο βαθμολόγησης και πολλά εργαλεία και βοηθητικές κλάσσεις για testing και επαλήθευση αποτελεσμάτων.
- **MLPNeuralNet** - Το MLPNeuralNet είναι μια γρήγορη βιβλιοθήκη multilayer perceptron για το iOS και το Mac OS X. Το MLPNeuralNet προβλέπει νέα παραδείγματα μέσω εκπαιδευμένων νευρωνικών δικτύων. Είναι χτισμένο πάνω στο Accelerate Framework της Apple χρησιμοποιώντας πράξεις δυανισμάτων και επιτάχυνση hardware(αν είναι διαθέσιμη).
- **MAChineLearning** - Μια βιβλιοθήκη multilayer perceptron για Objective-C, με πλήρη υποστήριξη για εκπαίδευση μέσω backpropagation. Εφαρμοσμένη με τη χρήση vDSP και vecLib, είναι 20 φορές ταχύτερη από την αντίστοιχη βιβλιοθήκη σε Java. Περιλαμβάνει δείγματα κώδικα για χρήση από το Swift.
- **BPN-NeuralNetwork** - Εφαρμογή νευρωνικού δικτύου τριών στρωμάτων(Στρώμα εισόδου, Κρυφό στρώμα, Στρώμα εξόδου) με χρήση Back Propagation. Μπορεί να χρησιμοποιηθεί για product recommendation systems, ανάλυση συμπεριφοράς χρηστών, εξόρυξη δεδομένων και αναλυση.
- **Multi-Perceptron-NeuralNetwork** - Εφαρμογή multi-perceptron με Back Propagation και άπειρα κρυφά στρώματα.
- **KRHebbian-Algorithm** - Αυτοδίδακτος αλγόριθμος χωρίς supervisor.
- **KRKmeans-Algorithm** - Αλγόριθμος K-Means για classification. Μπορεί να χρησιμοποιηθεί για εξόρυξη δεδομένων και συμπίεση εικόνας.
- **KRFuzzyCMeans-Algorithm** - Αλγόριθμος Fuzzy C-Means (FCM) για classification. Μπορεί να χρησιμοποιηθεί για εξόρυξη δεδομένων και συμπίεση εικόνας.

3.2.2.2 Swift

- **Bender** - Μια γρήγορη βιβλιοθήκη για την δημιουργία νευρωνικών δικτύων που είναι χτισμένη πάνω στο Metal της Apple. Υποστηρίζει μοντέλα Tensorflow.
- **Swift AI** - Το Swift AI είναι μια βιβλιοθήκη deep learning υψηλής απόδοσης γραμμένη εξολοκλήρου σε Swift. Υποστηρίζει όλες της πλατφόρμες της Apple και μελλοντικά θα έχει υποστήριξει και σε linux.
- **BrainCore** - Απλή αλλά γρήγορη βιβλιοθήκη για την δημιουργία νευρωνικών δικτύων που χρησιμοποιεί το Metal.
- **swix** - Στοχεύει στην ευκολία της μεταφοράς μοντέλων από Python/MATLAB σε Swift.
- **DeepLearningKit** - Ένα Framework ανοικτού κώδικα βαθιάς εκμάθησης(Deep Learning) για iOS, OS X και tvOS της Apple. Αυτή τη στιγμή επιτρέπει τη χρήση μοντέλων συνελικτικών νευρωνικών δικτύων που εκπαιδεύονται στο Caffe σε λειτουργικά συστήματα της Apple.
- **AIToolbox** - Μία εργαλειοθήκη από μοντέλα τεχνητής νοημοσύνης γραμμλενα σε Swift: Graphs/Trees, Linear Regression, Support Vector Machines, Neural Networks, PCA, KMeans, Genetic Algorithms, MDP, Mixture of Gaussians.
- **MLKit** - Μια απλή βιβλιοθήκη μηχανικής μάθησης γραμμένη σε Swift. Περιέχει εφαρμογές των Simple Linear Regression, Polynomial Regression και Ridge Regression.
- **Swift Brain** - Η πρώτη βιβλιοθήκη Μηχανικής Μάθησης γραμμένη σε Swift
- **Perfect Tensorflow** - Ενα wrapper API για το το Tensorflow C API γραμμενο σε Swift.
- **Awesome CoreML/Awesome Core ML Models** - Τα δύο μεγαλύτερα repos με προ-εκπαιδευμένα μοντέλα σε μορφή .mlmodel αρχείων για την άμεση χρήση τους σε ένα Xcode project.

3.2.3 Tensorflow support for Swift

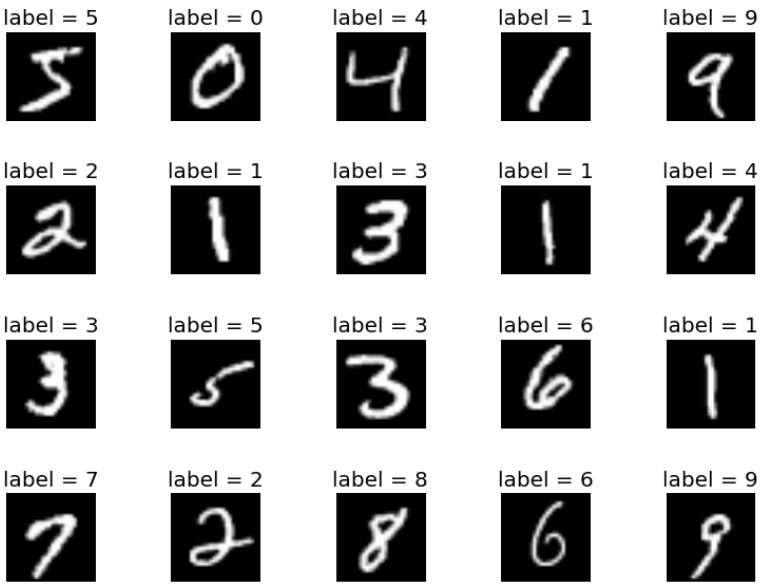
Η υποστήριξη Swift για Tensorflow παρέχει ένα νέο μοντέλο προγραμματισμού που συνδυάζει την απόδοση των γράφων(graphs) με την ευελιξία και τη σαφήνεια της δυναμικής εκτέλεσης(eager execution), με έντονη εστίαση στη βελτίωση της χρηστικότητας σε κάθε επίπεδο της στοίβας. Αυτό δεν είναι μόνο ένα Wrapper API του Tensorflow γραμμένο σε Swift - παρέχει βελτιώσεις στην γλώσσα και στον compiler της Swift για να προσφέρει μια εμπειρία πρώτης τάξης για τους προγραμματιστές μηχανικής μάθησης. Αυτο το project παρέχει ένα καινούργιο και διαφορετικό τρόπο για την χρήση του Tensorflow, ανοίγωντας καινούργιους τρόπους σχεδίασης για την λύση προβλημάτων. Το project είναι open source και υπάρχει ένα δημόσιο mailing list στο οποίο συζητούνται τα σχέδια του project και παρέχεται τεχνική βοήθεια.

ΚΕΦΑΛΑΙΟ 4 – ΣΥΝΟΛΑ ΔΕΔΟΜΕΝΩΝ ΕΙΚΟΝΩΝ

Τα σύνολα δεδομένων (datasets) είναι μεγάλου όγκου σύνολα συσχετίσεων ανάμεσα σε πρότυπα (patterns) και ετικέτες με τις περιγραφές τους (labels). Χρησιμοποιούνται από τους αλγορίθμους εκπαίδευσης μοντέλων στο στάδιο **training**, ώστε να μπορούν να αναγνωρίζουν πανομοιότυπα πρότυπα **εκτός του συνόλου** με το οποίο εκπαιδεύτηκαν και να τα αντιστοιχούν με τις σχετικές ετικέτες. Τα datasets χωρίζουν πάντα το σύνολο των συσχετίσεων τους σε δυο μέρη, ένα για training και ένα για testing – την διαδικασία από την οποία ελέγχεται η ακρίβεια με την οποία το μοντέλο μπορεί να προβλέψει. Συνήθως η αναλογία είναι από 70% training – 30% testing μέχρι 90% training – 10% testing.

4.1 ΑΝΑΓΝΩΡΙΣΗ ΧΑΡΑΚΤΗΡΩΝ (MNIST)

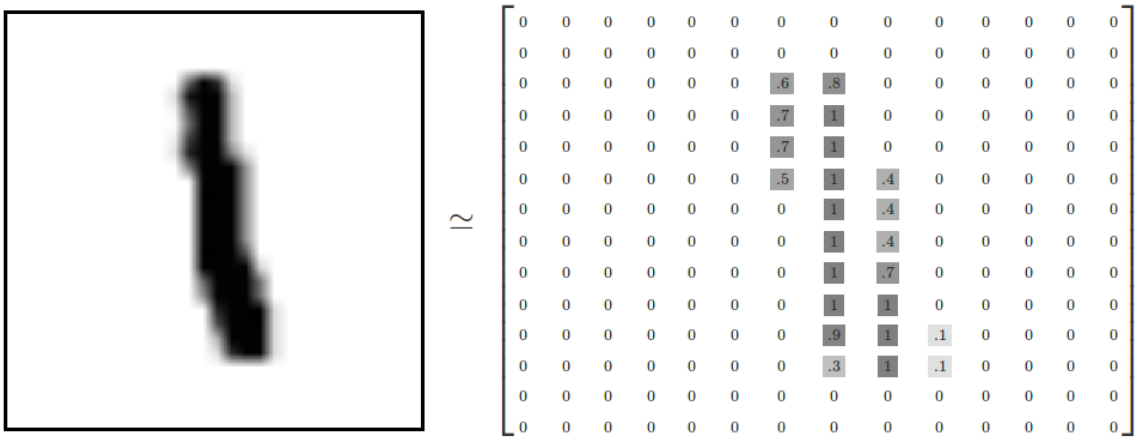
Ένα από τα πιο δημοφιλή σύνολα δεδομένων για εκπαίδευση μοντέλων της μηχανικής μάθησης είναι το **MNIST**, το οποίο έχουν συγκεντρώσει και δημιουργήσει ο Yann LeCun και ένας ερευνητής της Microsoft & Google Labs. Πρόκειται για μια βάση δεδομένων χειρόγραφων ψηφίων, με ένα σετ εκπαίδευσης των 55.000 παραδειγμάτων (training set) και ένα σετ δοκιμών 5.000 παραδειγμάτων (test set). Αυτό το σετ κατάρτισης έχει απομονωθεί από ένα μεγαλύτερο σετ που διατίθεται από την ειδική βάση δεδομένων **NIST 19**, η οποία περιέχει χειρόγραφα ψηφία, κεφαλαία και πεζά γράμματα. Ενδείκνυται ως καλή βάση δεδομένων για όσους θέλουν να δοκιμάσουν τεχνικές μάθησης και μεθόδους αναγνώρισης προτύπων σε δεδομένα του πραγματικού κόσμου, ενώ παράλληλα απαιτούνται ελάχιστες εώς μηδενικές προσπάθειες για επεξεργασία και μορφοποίηση. Το MNIST έχει γίνει ένα πρότυπο σημείο αναφοράς για συστήματα μάθησης, ταξινόμησης και οράσεως υπολογιστών.



Εικόνα 24: Ψηφία του MNIST & labels

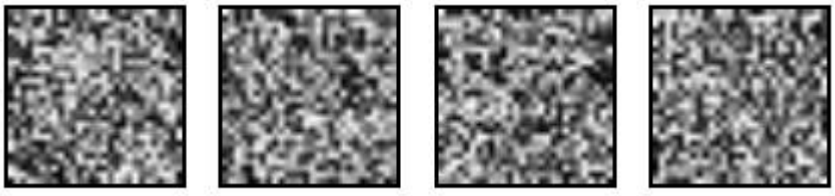
Όπως αναφέραμε νωρίτερα για τα σύνολα δεδομένων, κάθε «σημείο» του MNIST αποτελείται από 2 μέρη: μια εικόνα του χειρόγραφου ψηφίου και μια αντίστοιχη ετικέτα. Αποκαλούμε τις εικόνες «x» και τις ετικέτες «y».

Κάθε εικόνα έχει μέγεθος 28x28 pixels, κάτι που αναλύουμε σε έναν μεγάλο δισδιάστατο πίνακα αριθμών:



Εικόνα 25: Μετάφραση μια εικόνας του MNIST σε δισδιάστατο πίνακα

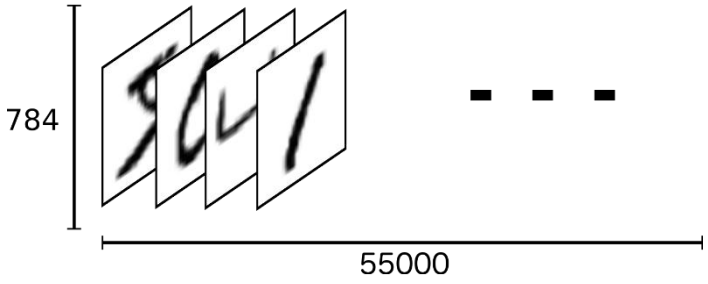
Μπορούμε να ευθειάσουμε τον πίνακα σε ένα διάνυσμα μεγέθους 28x28 = 784. Κάθε στοιχείο του πίνακα/διανύσματος περιέχει μια τιμή ανάμεσα στο 0 και το 1, που αναπαριστά την πυκνότητα του αντίστοιχου εικονοστοιχείου. Από αυτή την οπτική αντιμετωπίζουμε το MNIST σαν να είναι μια συλλογή από διανύσματα μεγέθους 784. Δεν είναι όλα τα στοιχεία σε αυτό τον 784 διαστάσεων χώρο ψηφία MNIST. Τυπικά τα σημεία σε αυτόν τον χώρο είναι πολύ διαφορετικά. Για να το αντιληφθούμε, μπορούμε να διαλέξουμε τυχαία μερικά σημεία και να τα εξετάσουμε. Σε ένα τυχαίο σημείο - μια τυχαία εικόνα 28x28 - κάθε εικονοστοιχείο είναι τυχαία μαύρο, λευκό ή κάποια απόχρωση του γκρι. Το αποτέλεσμα είναι ότι τα τυχαία σημεία φαίνονται σαν θόρυβος – μια αόριστη μορφή σε σχέση με τα χειρόγραφα ψηφία που περιμένουμε.



Εικόνα 26: Στοιχεία από τα διανύσματα μεγέθους 784

Στην πραγματικότητα η μετατροπή των δεδομένων που προαναφέραμε απομακρύνει πληροφορίες σχετικά με τη 2D δομή της εικόνας. Οι μέθοδοι ψηφιακής όρασης στον υπολογιστή εκμεταλλεύονται αυτή τη δομή, αλλά όχι η απλή μέθοδος «regression softmax» που χρησιμοποιούμε.

Το αποτέλεσμα είναι ένα **tensor** που βρίσκεται στο αντικείμενο `mnist.train.images` και είναι στην πράξη ένας πίνακας n διαστάσεων με την μορφή [55000, 784]. Η πρώτη διάσταση είναι αναφορά στην σειρά των εικόνων και η δεύτερη διάσταση αναφορά στο κάθε εικονοστοιχείο, για κάθε εικόνα. Κάθε στοιχείο του tensor είναι η ένταση ενός εικονοστοιχείου ανάμεσα σε 0 και 1, για κάποιο συγκεκριμένο εικονοστοιχείο της εικόνας.



Εικόνα 27: Διάνυσμα [55000, 784] - Οι MNIST εικόνες με τα pixel τους

Όπως είδαμε νωρίτερα, κάθε εικόνα στο MNIST έχει μια αντιστοίχιση με μια ετικέτα, έναν αριθμό [0 - 9] που αναπαριστά το χειρόγραφο ψηφίο της εικόνας. Αυτές τις ετικέτες τις αποκαλούμε «one-hot vectors». Κάθε μια από αυτές έχει 0 στις πιο πολλές θέσεις, και 1 σε μια θέση. Σε αυτή την περίπτωση, το t ψηφίο αναπαρίσταται ως ένα διάνυσμα που είναι 1 στην t διάσταση. Για παράδειγμα, ο αριθμός 3 θα ήταν μια αναπαράσταση [0, 0, 0, **1**, 0, 0, 0, 0, 0, 0]. Συνεπώς το σύνολο labels του MNIST είναι μια σειρά από [55000, 10] αριθμούς.

Εφαρμογή ταξινομητή Softmax σε MNIST (regression)

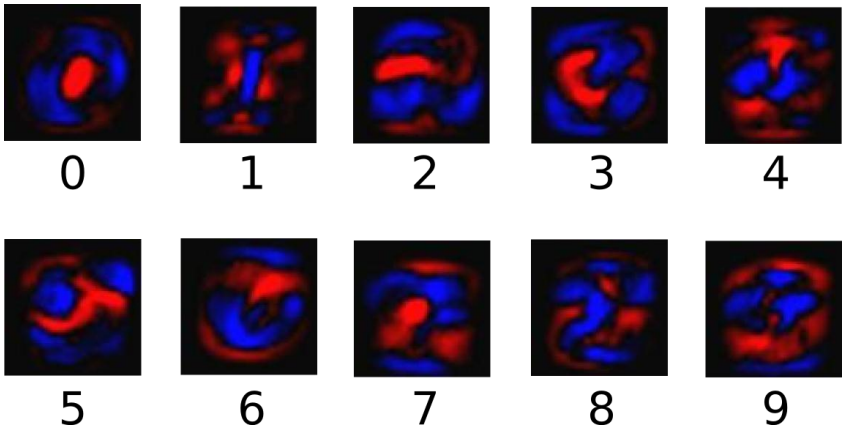
Γνωρίζουμε οτι κάθε εικόνα του MNIST είναι ένα χειρόγραφο ψηφίο ανάμεσα σε 0 και 9. Συνεπώς αυτή η εικόνα μπορεί να δείχνει 10 πιθανά πράγματα. Θέλουμε δοθείσης της εικόνας να μπορούμε να απαντήσουμε για τις **πιθανότητες** παράστασης κάθε ψηφίου. Για παράδειγμα, μπορεί το μοντέλο να πάρει μια εικόνα του ψηφίου 9, και να είναι σίγουρο κατα 80% οτι βλέπει το 9, αλλά ταυτόχρονα έχει 5% πιθανότητα να είναι 8 (λόγω του κύκλου στο επάνω μέρος) και 5% πιθανότητα να είναι όλα τα άλλα ψηφία – δεν είναι 100% σίγουρο.

Αυτή είναι μια κλασσική περίπτωση όπου η softmax regression αποτελεί ένα απλό μοντέλο. Αν θέλουμε να αντιστοιχίσουμε πιθανότητες σε ένα αντικείμενο ανάμεσα σε πολλά διαφορετικά, αυτό που χρειαζόμαστε είναι η softmax, γιατί μας δίνει μια λίστα τιμών ανάμεσα στο 0 και το 1, που αθροίζονται σε 1. Ακόμα και σε δυσκολότερες περιπτώσεις και πιο περίπλοκα μοντέλα, το τελευταίο στρώμα είναι μια συνάρτηση softmax.

Η παλινδρόμηση softmax έχει δύο βήματα: πρώτα προσθέτουμε τα στοιχεία εισόδου μας σε ορισμένες κατηγορίες, και στη συνέχεια μετατρέπουμε αυτά τα στοιχεία σε πιθανότητες.

Για να συγκεντρώσουμε τα στοιχεία που αποδεικνύουν ότι μια δεδομένη εικόνα ανήκει σε μια συγκεκριμένη κλάση, κάνουμε ένα σταθμισμένο άθροισμα των εντάσεων των εικονοστοιχείων. Το βάρος είναι αρνητικό εάν το εικονοστοιχείο που έχει υψηλή ένταση δεν βρίσκεται σε αυτή την κατηγορία και θετικό αν υπάρχουν αποδεικτικά στοιχεία υπέρ του (πιθανό να βρίσκεται στην κατηγορία).

Το παρακάτω διάγραμμα δείχνει τα βάρη ενός μοντέλου που είδαμε για κάθε μία από αυτές τις κατηγορίες. Το κόκκινο αντιπροσωπεύει τα αρνητικά βάρη, ενώ το μπλε αντιπροσωπεύει θετικά βάρη.



Εικόνα 28: Βάρη ενός μοντέλου MNIST

Προσθέτουμε επίσης κάποια επιπλέον στοιχεία που ονομάζονται κλίσεις (biases). Για την ακρίβεια θέλουμε να είμαστε σε θέση να πούμε ότι ορισμένα πράγματα είναι πιθανότερο να είναι ανεξάρτητα από την εισροή. Το αποτέλεσμα είναι ότι τα αποδεικτικά στοιχεία μιας κατηγορίας που δίνεται σε μια εισροή x είναι:

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i$$

Όπου W_i είναι τα βάρη, b_i οι κλίσεις για την ΄κλάση i, και j είναι ένας δείκτης για το άθροισμα των εικονοστοιχείων της εικόνας x. Στη συνέχεια μετατρέπουμε τα στοιχεία σε πιθανές προβλέψεις χρησιμοποιώντας την συνάρτηση softmax:

$$y = \text{softmax}(\text{evidence})$$

Η softmax χρησιμεύει ως συνάρτηση "**ενεργοποίησης**" ή "σύνδεσης", διαμορφώνει δηλαδή την έξοδο της γραμμικής μας λειτουργίας στη μορφή που θέλουμε - στην περίπτωση αυτή, μια κατανομή πιθανότητας σε 10 περιπτώσεις. Πρόκειται με απλά λόγια για μετατροπή των στοιχείων της απόδειξης (evidence) σε πιθανότητες της εισροής μας να ανήκει σε κάποια κατηγορία. Ορίζεται ως:

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

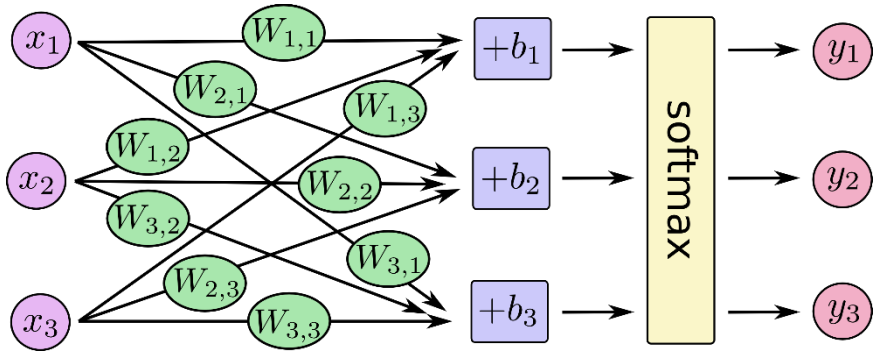
Και αν επεκτείνουμε την εξίσωση, παίρνουμε

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Είναι συχνά πιο χρήσιμο να σκεφτούμε την softmax με τον πρώτο τρόπο: υψώνοντας τις εισόδους του σε δύναμη και στη συνέχεια «κανονικοποιώντας» τις. Η ύψωση σε δύναμη σημαίνει ότι μία ακόμη μονάδα ενδείξεων αυξάνει το βάρος που δίνεται σε οποιαδήποτε υπόθεση πολλαπλάσια.

Και αντιστρόφως, η ύπαρξη μιας μικρότερης μονάδας αποδεικτικών στοιχείων (οτι ανήκει σε μια κατηγορία) σημαίνει ότι μια υπόθεση παίρνει ένα κλάσμα του παλαιότερου βάρους της. Καμία υπόθεση δεν έχει ποτέ μηδενικό ή αρνητικό βάρος. Στη συνέχεια, η Softmax ομαλοποιεί αυτά τα βάρη, ώστε να προσθέσουν μέχρι ένα, σχηματίζοντας μια έγκυρη κατανομή πιθανότητας.

Η συνάρτηση softmax μπορεί να απεικονιστεί όπως παρακάτω, αν και στην πραγματικότητα λειτουργεί με πολλά περισσότερα x. Για κάθε έξοδο, υπολογίζουμε ένα σταθμισμένο άθροισμα του x, προσθέτουμε μια κλιση (b) και στη συνέχεια εφαρμόζουμε softmax.



Εικόνα 29: Συνάρτηση Softmax

Και αν υπολογίσουμε τις εξισώσεις παίρνουμε:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

Για ευκολία υπολογισμών μετατρέουμε τα παραπάνω σε διανύσματα και προκύπτει πολλαπλασιαμσός πινάκων και άθροισμα διανύσματος:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Ή πιο απλά:

$$y = \text{softmax}(Wx + b)$$

4.2 ΑΛΛΑ ΣΥΝΟΛΑ ΔΕΔΟΜΕΝΩΝ

Είναι προφανές από την καθημερινότητα και τις χρήσεις της ΜΜ σε αυτήν, οτι δεν περιοριζόμαστε σε προβλέψεις και ταξινόμηση αριθμών και χειρόγραφων ψηφίων ή ψηφίων γενικότερα. Στην πράξη υπάρχουν πολλά περισσότερα αντικείμενα που μπορεί να ταξινομήσει ένα μοντέλο, και με πολύ πολυπλοκότερες δομές – συνεπώς χρειαζόμαστε και άλλα σύνολα δεδομένων για να εκπαιδεύσουμε τα μοντέλα αυτά.

MS-COCO

Το COCO (Common Objects in Context) είναι ένα σύνολο δεδομένων μεγάλης κλίμακας και αρκετά πλούσιο για αναγνώριση, κατηγοριοποίηση και καταγραφή αντικειμένων. Μπορεί να αναγνωρίσει αντικείμενα, να τα εντοπίσει μέσα στο περιβάλλον. Έχει 330.000 εικόνες με 200.000 ετικέτες, που απεικονίζουν 1.500.000 αντικείμενα, από 80 κατηγορίες. Το μέγεθός του δεν ξεπερνάει τα 25GB (σε συμπιεσμένη μορφή).

Εικόνα 30: Παραδείγματα αντικειμένων που αναγνωρίζει το COCO

ImageNet

Το ImageNet είναι ένα σύνολο δεδομένων εικόνων που οργανώνονται σύμφωνα με την ιεραρχία του WordNet. Το WordNet περιέχει περίπου 100.000 φράσεις και το ImageNet παρέχει περίπου 1000 εικόνες κατά μέσον όρο για να απεικονίσει κάθε φράση. Περιέχει πάνω από 20 χιλιάδες διφορούμενες κατηγορίες, ενώ μια τυπική κατηγορία όπως "μπαλόνι" ή "φράουλα" περιέχει αρκετές εκατοντάδες εικόνες. Η βάση δεδομένων των διευθύνσεων URL εικόνων τρίτων είναι ελεύθερα διαθέσιμη απευθείας από το ImageNet, ωστόσο, οι πραγματικές εικόνες δεν ανήκουν στο ImageNet.

Εικόνα 31: Παραδείγματα που αναγνωρίζει το ImageNet

Open Images Dataset

Αποτελείται από περίπου 9 εκατομμύρια διευθύνσεις εικόνων. Αυτές οι εικόνες έχουν σημειωθεί με ετικέτες που οριοθετούν «κουτιά» στα οπία εκτείνονται σε χιλιάδες κλάσεις. Το σύνολο δεδομένων περιλαμβάνει ένα σύνολο εκπαίδευσης 9.011.219 εικόνων, ένα σετ επικύρωσης 41.260 εικόνων και ένα σύνολο δοκιμών 125.436 εικόνων, ενώ διαθέτει περίπου 5.000 ετικέτες.

SVHN (Street View House Numbers)

Πρόκειται για ένα σύνολο δεδομένων του πραγματικού κόσμου για την ανάπτυξη αλγορίθμων ανίχνευσης αντικειμένων. Είναι παρόμοιο με το σύνολο δεδομένων MNIST που αναφέραμε εκτενώς νωρίτερα, αλλά έχει περισσότερες ετικέτες δεδομένων - πάνω από 600.000 εικόνες. Τα δεδομένα έχουν συλλεχθεί από τους αριθμούς κατοικιών που εμφανίζονται στην προβολή Google Street View.

Εικόνα 32: Εικόνες που περιέχονται στο SVHN

CIFAR-10

Άλλο ένα σύνολο δεδομένων για ταξινόμηση εικόνων. Αποτελείται από 60.000 εικόνες από 10 τάξεις (κάθε τάξη αντιπροσωπεύεται ως μια σειρά στην παρακάτω εικόνα). Συνολικά, υπάρχουν 50.000 εικόνες εκπαίδευσης και 10.000 εικόνες δοκιμών. Το σύνολο δεδομένων χωρίζεται σε 6 μέρη - 5 σετ εκπαίδευσης και 1 σετ δοκιμής. Κάθε σετ έχει 10.000 εικόνες.

Εικόνα 33: Οι 10 τάξεις του CIFAR-10

ΚΕΦΑΛΑΙΟ 5

ΕΦΑΡΜΟΓΗ «DISTRIBUTED MNIST»

Όπως ήδη αναφέραμε, το MNIST Dataset είναι ένα σύνολο δεδομένων χειρόγραφων αριθμών (0-9) το οποίο χρησιμοποιείται πολύ συχνά για να δοκιμαστεί ένας αλγόριθμος μηχανικής μάθησης. Οι περισσότεροι γνώστες του αντικειμένου, αναφέρουν το MNIST ως το “Hello World” της μηχανικής μάθησης, μια πολύ απλή μορφή – δοκιμή αλγορίθμων. Μπορεί να χρησιμοποιηθεί με πολλές διαφορετικές αρχιτεκτονικές δικτύων, από πολύ γνωστά και πολύπλοκα μοντέλα όπως το AlexNet, μέχρι απλά δίκτυα ενός επιπέδου. Ενδεικτικά παρατίθενται μερικές αναφορές απόδοσης (ποσοστό σφάλματος) πολλών δικτύων με την χρήση του MNIST από την ιστοσελίδα του Yann LeCun, γνωστού μηχανικού πληροφορικής και πλέον ειδικού σε συνελικτικά νευρωνικά δίκτυα και οπτική αναγνώριση http://yann.lecun.com/exdb/mnist/ :

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN)	none	12.0	LeCun et al. 1998
pairwise linear classifier	deskewing	7.6	LeCun et al. 1998
K-Nearest Neighbors			
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
Non-Linear Classifiers			
40 PCA + quadratic classifier	none	3.3	LeCun et al. 1998
Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
Convolutional nets			

Convolutional net LeNet-5, [distortions]	none	0.8	LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	none	0.7	LeCun et al. 1998
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +-0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

5.1 – ΕΚΠΑΙΔΕΥΣΗ

Στην επίσημη σελίδα του Tensorflow, προσφέρονται πολλά παραδείγματα, σε γλώσσες Python, Java και Go, για πολλές περιπτώσεις χρήσης, όπως το python εκτελέσιμο «deep_mnist.py» το οποίο χαρακτηρίζεται ως ένα «deep MNIST classifier» με χρήση συνελικτικών επιπέδων (using convolutional layers).

Η διαδικασία εκπαίδευσης γίνεται από την deepnn() όπως παρακάτω:

def deepnn(x):

```
# Η μεταβλητή x είναι ένας tensor δυο διαστάσεων σχήματος [None, 784] οπου το
784 αναπαριστά μία flattened εικόνα 28 επί 28 pixel και το None σημαίνει
οποιοδήποτε διάσταση, αυτο γίνετε ώστε να μπορεί να χωριστεί το x σε batches.
Επισης είναι δηλωμένο σαν tf.Placeholder, τα tf.Variables δεν μπορούν να έχουν
μεταβλητές διαστάσεις.

# Εδώ γίνεται reshape το x από 2d tensor σε 4d tensor όπου η πρώτη διάσταση
είναι dummy και ορίζεται implicitly γιαυτό και είναι -1, ο λογος που υπάρχει
είναι γιατί το x θα πολλαπλασιαστεί με το W το οποίο είναι 4d. Η δεύτερη και η
τρίτη είναι μήκος κ πλάτος της εικόνα 28x28 και η τέταρτη ορίζει πόσα κανάλια
χρώματος έχει η εικόνα.

  x_image = tf.reshape(x, [-1, 28, 28, 1])

#Εδώ ορίζεται το πρώτο συνελικτικό στρώμα μαζί με το max pooling στο τέλος του.
  W_conv1 = weight_variable([5, 5, 1, 32])
  b_conv1 = bias_variable([32])
  h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
  h_pool1 = max_pool_2x2(h_conv1)
#Εδώ ορίζεται το δεύτερο συνελικτικό στρώμα μαζί με το max pooling στο τέλος
του.
```

```
  W_conv2 = weight_variable([5, 5, 32, 64])
  b_conv2 = bias_variable([64])
  h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
  h_pool2 = max_pool_2x2(h_conv2)


#Εδώ ορίζεται το πρώτο fully connected στρώμα
  W_fc1 = weight_variable([7 * 7 * 64, 1024])
  b_fc1 = bias_variable([1024])

  h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
  h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#Εδώ ορίζεται το dropout και η πιθανότητα/συχνότητα του
  keep_prob = tf.placeholder(tf.float32)
  h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#Εδώ ορίζεται το δεύτερο fully connected στρώμα
  W_fc2 = weight_variable([1024, 10])
  b_fc2 = bias_variable([10])

#Τελος επιστρέφεται το τελικό στρώμα
  y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
  return y_conv, keep_prob
```

Ο παραπάνω κώδικας εκτελείται είτε σε ένα μηχάνημα (πχ πυρήνες μιας CPU / GPU) είτε κατανεμημένα σε πολλά μηχανήματα μέσω δικτύου. Προκειμένου να κατανεμηθούν οι εργασίες σε περισσότερα από ένα μηχανήματα (parameter server, workers) χρησιμοποιούμε τις παραμέτρους ps_hosts & worker_hosts στα command line arguments ώστε να ενημερώσουμε το αντικείμενο ClusterSpec για τις διευθύνσεις των μονάδων που αναλαμβάνουν χρέη parameter servers και workers αντίστοιχα. Αυτό επιτυγχάνεται με την διαδικασία main(_) όπως παρακάτω:

```
def main(_):

# «Διάβασμα» παραμέτρων ps_hosts και worker_hosts (οι τιμές τους χωρίζονται με
κομα)

  ps_hosts = FLAGS.ps_hosts.split(",")
  worker_hosts = FLAGS.worker_hosts.split(",")

  # Δημιουργία του ClusterSpec με τα hosts που ορίζουν τα command line
arguments.
  cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})


  # Κατανομή της μνήμης των γραφικών
  config = tf.ConfigProto()
  config.gpu_options.per_process_gpu_memory_fraction = 0.3
   # Δημιουργία Server αντικειμένου για τοπική διεργασία (αναλόγως ps/worker
και index)
```

```
  server = tf.train.Server(cluster, job_name=FLAGS.job_name,
  task_index=FLAGS.task_index)

  if FLAGS.job_name == "ps":
       server.join()
  elif FLAGS.job_name == "worker":

       with tf.device(tf.train.replica_device_setter(
         worker_device="/job:worker/task:%d" % FLAGS.task_index,
       cluster=cluster)):
      # Είσοδος δεδομένων
       mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)


       # Δημιουργία του μοντέλου
       x = tf.placeholder(tf.float32, [None, 784])
       y_ = tf.placeholder(tf.float32, [None, 10])
       y_conv, keep_prob = deepnn(x)
      cross_entropy = tf.reduce_mean(
          tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))

       global_step = tf.contrib.framework.get_or_create_global_step()

       train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy,
lobal_step=global_step)

       correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
      accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

       # Η StopAtStepHook ρυθμίζει τον μέγιστο αριθμό επαναλήψεων (σε 1000)
    hooks=[tf.train.StopAtStepHook(last_step=1000)]
 with tf.train.MonitoredTrainingSession(master=server.target,
                                        is_chief=(FLAGS.task_index == 0),
                                        checkpoint_dir=FLAGS.log_dir,
                                        hooks=hooks) as mon_sess:
       i = 0
       while not mon_sess.should_stop():

       # Ασύγχρονη εκπαίδευση
       batch = mnist.train.next_batch(50)
       if i % 100 == 0:
       train_accuracy = mon_sess.run(accuracy, feed_dict={
              x: batch[0], y_: batch[1], keep_prob: 1.0})

       print('global_step %s, task:%d_step %d, training accuracy %g'
              % (tf.train.global_step(mon_sess, global_step),
FLAGS.task_index, i, train_accuracy))

       mon_sess.run(train_step, feed_dict={x: batch[0], y_: batch[1], keep_prob:
0.5})
```

```
        i = i + 1

export_path = '/home/user/Tensorflow_models/'
builder = tf.saved_model.builder.SavedModelBuilder(export_path)



builder.add_meta_graph_and_variables(mon_sess,
                                     [tag_constants.TRAINING],
                                     signature_def_map=foo_signatures,
                                     assets_collection=foo_assets,
                                  strip_default_attrs=True)
builder.save()
```

Ο παραπάνω κώδικας θα εκτελεστεί σε διαφορετικά μηχανήματα (για παράδειγμα σε ένα δίκτυο LAN 10.0.0.0/24) ως εξής:

Parameter Server:

```
$ python3.6 distributed-deep-mnist-with-queue.py \
--ps_hosts=10.0.0.1:2222 \
--worker_hosts=10.0.0.2:2222, 10.0.0.3:2222 \
--job_name=ps --task_index=0 –log_dir=/home/user/tensorflow/tmp/
```

Worker #1:

```
$ python3.6 distributed-deep-mnist-with-queue.py \
--ps_hosts=10.0.0.1:2222 \
--worker_hosts=10.0.0.2:2222, 10.0.0.3:2222 \
--job_name=worker --task_index=0 –log_dir=/home/user/tensorflow/tmp/
```

Worker #2:

```
$ python3.6 distributed-deep-mnist-with-queue.py \
--ps_hosts=10.0.0.1:2222 \
--worker_hosts=10.0.0.2:2222, 10.0.0.3:2222 \
--job_name=worker --task_index=1 –log_dir=/home/user/tensorflow/tmp/
```

Έτσι το μοντέλο εκπαιδεύουν 2 ***workers*** *(Το μηχάνημα στην διεύθυνση 10.0.0.2 – worker 1, και το μηχάνημα worker 2 στην διεύθυνση 10.0.0.3) ενώ τις τιμές των παραμέτρων του γράφου «κρατάει» ο* ***parameter server*** *(Το μηχάνημα στην διεύθυνση 10.0.0.1).*

5.2 – ΕΚΤΕΛΕΣΗ ΣΕ ΚΙΝΗΤΕΣ ΣΥΣΚΕΥΕΣ

Μετά την επιτυχημένη εκπαίδευση του μοντέλου, εξάγουμε τον γράφο στο αρχείο mnist.pb και τις ετικέτες (labels) των αριθμών στο mnist_labels.txt αρχείο. Ανάλογα με την πλατφόρμα στην οποία θα χρησιμοποιηθεί (Android / iOS) και με ποια έκδοση του tensorflow (Tensorflow for Mobiles / Tensorflow Lite), το αρχείο του μοντέλου πιθανόν να χρειάζεται μετατροπή (FreezeGraph). Στην περίπτωση που το μοντέλο θα χρησιμοποιηθεί σε Lite έκδοση, χρειάζεται απαραίτητα το αρχείο checkpoint (mnist.ckpt) κατα την επεξεργασία με το **Graph Transform Tool.**

5.2.1 ANDROID

Προκειμένου να χρησιμοποιηθούν τα εκπαιδευμένα μοντέλα στο Tensorflow mobile, χρειάζεται η απαραίτητη προετοιμασία του Android NDK, ώστε να μπορεί το Android Studio (IDE) να μεταγλωττίσει το tensorflow σε εκτελέσιμη για το android μορφή βιβλιοθήκης (*.so). Τα βασικά βήματα για την εγκατάσταση του NDK είναι η ληψη του *.zip αρχείου (η τελευτάια έκδοση stable είναι η r17 - android-ndk-r17-linux-x86_64.zip). Μετά την εξαγωγή του αρχείου zip, οι εντολές

```
$ export $NDK_HOME =”/home/user/android-ndk-r17”
$ export PATH=$PATH:$NDK_HOME
$ cd jni-build
$ make
$ make install
```

θα μεταγλωττίσουν και θα δημιουργήσουν το αρχείο *.so, που αποτελεί την χρήσιμη πλέον για το android studio βιβλιοθήκη του tensorflow. Για να χρησιμοποιήσουμε το *.so αρχείο το αντιγράφουμε στο Android Studio Project, στην θέση app/src/main/jniLibs/armeabi-v7a/.

Τα βήματα μέχρι αυτό το σημείο με την πάροδο του χρόνου αντικαταστάθηκαν με απλούστερα, όπως την χρήση της tensorflow βιβλιοθήκης απευθείας με dependency από το Gradle Build file του project – όπως χρησιμοποιούνται οι περισσότερες βιβλιοθήκες σε android. Παρ’ όλα αυτά, ο προηγούμενος τρόπος είναι διαθέσιμος και χρησιμοποιείται στις περιπτώσεις που κάποιος θέλει να ξαναμεταγλωττίσει το tensorflow for mobile “from source”.

Η παραπάνω διαδικασία μπορεί να απλοποιηθεί με την χρήση ενός dependency όπως παρακάτω, στο gradle.build αρχείο του android project:

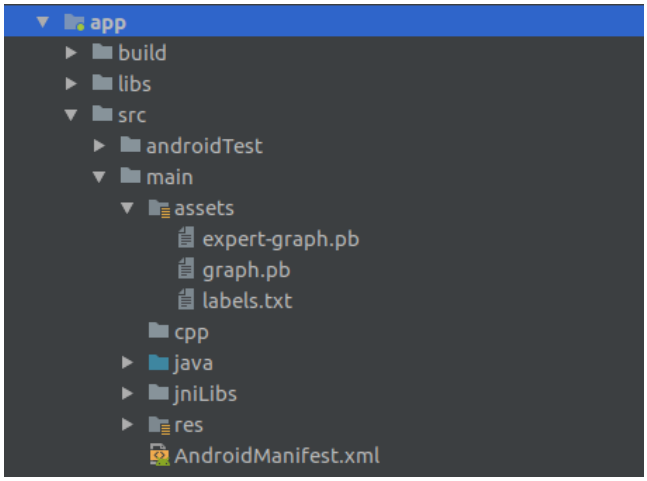
```
allprojects {
    repositories {
        jcenter()
    }
}
dependencies {
    compile 'org.tensorflow:tensorflow-android:+'
}
```

Δημιουργία MNIST Demo app

Για να οπτικοποιήσουμε και να παρουσιάσουμε τα αποτελέσματα του MNIST (που εκπαιδεύσαμε κατανεμημένα στο κεφ. 5.1) σε μια εφαρμογή android, χρησιμοποιήσαμε το UI element "DrawView" (`public class DrawView extends View`) στο οποίο ο χρήστης μπορεί να «ζωγραφίσει» στην οθόνη ένα σχήμα, για το οποίο προγραμματιστικά έχουμε πρόσβαση στα pixels. Εναλλακτικά θα μπορούσαμε να χρησιμοποιήσουμε φωτογραφίες (ίσως από την κάμερα της συσκευής) μειώνοντας το μέγεθος στα απαραίτητα εικονοστοιχεία (28x28 pixels).

```
DrawView  drawView = (DrawView) findViewById(R.id.draw);
drawModel = new DrawModel(PIXEL_WIDTH, PIXEL_WIDTH);
drawView.setModel(drawModel);
```

Η κλάση Classifier δέχεται το «σκίτσο» του χρήστη και το εισάγει στο tensorflow μοντέλο. Χρησιμοποιεί το .pb αρχείο που τοποθετήσαμε στον φάκελο assets, και αντιστοιχίζει τις ετικέτες που βρίσκει στο αρχείο labels.txt στον ίδιο φάκελο του project.



Εικόνα 34: Αρχεία *.pb & *.txt στον φάκελο android assets

```
public class Classifier {

        // Only returns if at least this confidence
        private static final float THRESHOLD = 0.1f;
        private TensorFlowInferenceInterface tfHelper;
        private String inputName;
        private String outputName;
        private int inputSize;
        private List<String> labels;
        private float[] output;
        private String[] outputNames;
        static private List<String> readLabels(Classifier c, AssetManager am,
    String fileName) throws IOException {
            BufferedReader br = null;
            br = new BufferedReader(new InputStreamReader(am.open(fileName)));
            String line;
            List<String> labels = new ArrayList<>();
```

```
        while ((line = br.readLine()) != null) {
            labels.add(line);
        }
        br.close();
        return labels;
    }

    static public Classifier create(AssetManager assetManager, String
modelPath, String labelPath, int inputSize, String inputName, String
outputName)
                            throws IOException {
        Classifier c = new Classifier();
        c.inputName = inputName;
        c.outputName = outputName;
        // Read labels
        String labelFile = labelPath.split("file:///android_asset/")[1];
        c.labels = readLabels(c, assetManager, labelFile);
        c.tfHelper = new TensorFlowInferenceInterface();
        if (c.tfHelper.initializeTensorFlow(assetManager, modelPath) != 0) {
            throw new RuntimeException("TF initialization failed");
        }
        int numClasses = 10;
        c.inputSize = inputSize;
        // Pre-allocate buffer.
        c.outputNames = new String[]{ outputName };
        c.outputName = outputName;
        c.output = new float[numClasses];
        return c;
    }

    public Classification recognize(final float[] pixels) {
        tfHelper.fillNodeFloat(inputName, new int[]{inputSize * inputSize},
pixels);
        tfHelper.runInference(outputNames);
        tfHelper.readNodeFloat(outputName, output);
        // Find the best classification
        Classification ans = new Classification();
        for (int i = 0; i < output.length; ++i) {
            System.out.println(output[i]);
            System.out.println(labels.get(i));
            if (output[i] > THRESHOLD && output[i] > ans.getConf()) {
                ans.update(output[i], labels.get(i));
            }
        }
        return ans;
    }
}
```

Δημιουργούμε ένα αντικείμενο της κλάσης Classifier με την χρήση της Classifier.create() ώς εξής:

```
try {
```

```
 classifier = Classifier.create(getApplicationContext().getAssets(),
                                    MODEL_FILE,
                                    LABEL_FILE,
                                    INPUT_SIZE,
                                    INPUT_NAME,
                                    OUTPUT_NAME);
} catch (final Exception e) {
    throw new RuntimeException("Error initializing TensorFlow!", e);
}
```

και εισάγουμε τα pixels που ζωγράφισε ο χρήστης στον classifier:

```
//float array που περιέχει τα ζωγραφισμένα pixels
    float pixels[] = drawView.getPixelData();
    final Classification res = classifier.recognize(pixels);
    String result = "Result: ";
    if (res.getLabel() == null) { resText.setText(result + "?"); }
       //Η res.getLabel() επιστρέφει την αντίστοιχη ετικέτα από το
labels.txt αρχείο
    else {
        result += res.getLabel();
        result += "\nwith probability: " + res.getConf();
        resText.setText(result);
    }
```

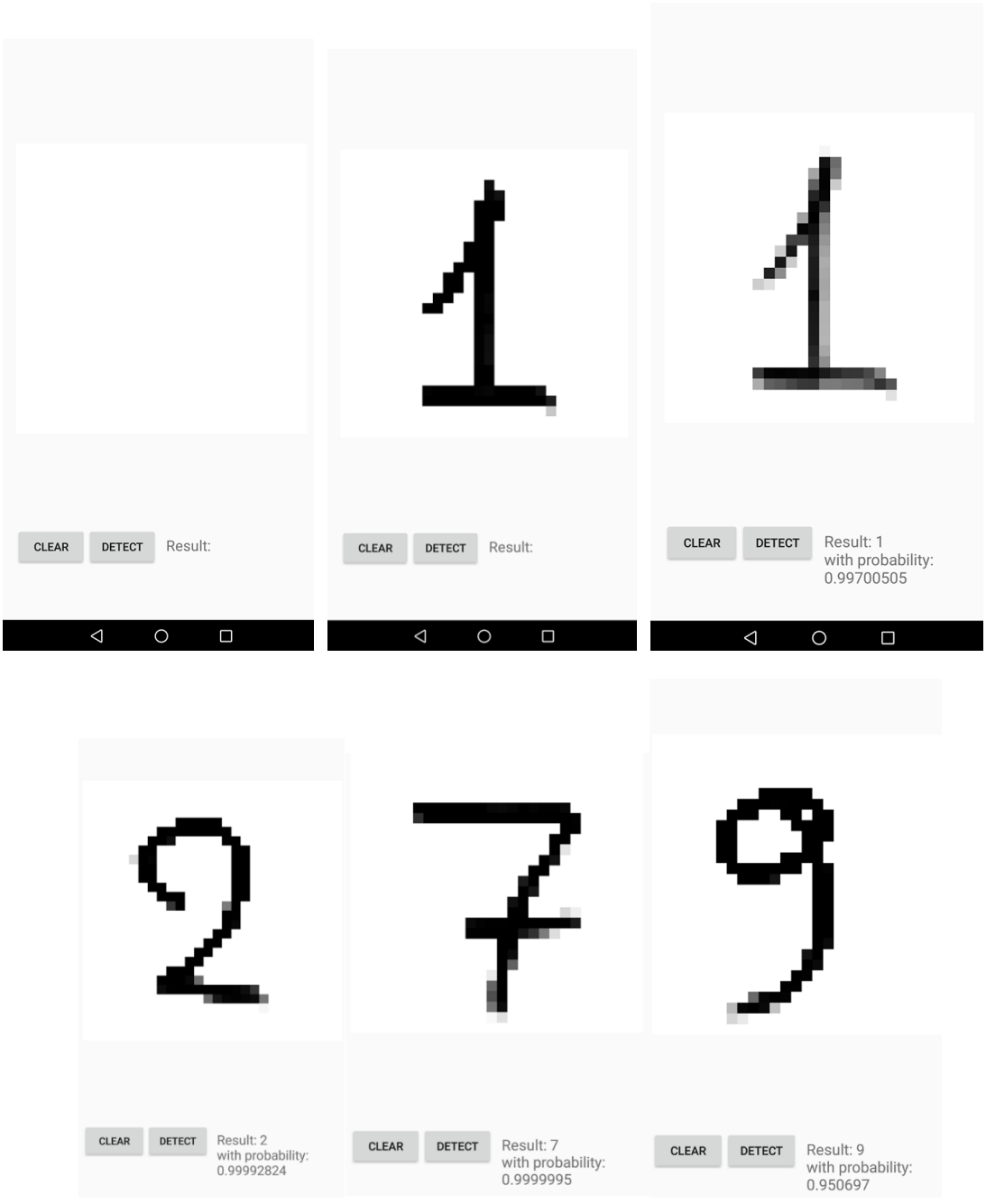
Για να δημιουργήσουμε το εκτελέσιμο του android (*.apk) χρησιμοποιούμε το Bazel build tool (εξ ορισμού build tool του android studio είναι το Gradle – Android SDK + Java) λόγω της χρήσης του Android NDK δηλαδή κώδικα C++:

```
$ bazel build -c opt //mnist/examples/android:mnist _demo
```

και στην συνέχεια για να το εγκαταστήσουμε σε μια σύσκευή μέσω USB, χρησιμοποιούμε το command line ADB:

```
$ adb install -r bazel-bin/ mnist/examples /android/mnist_demo.apk
```

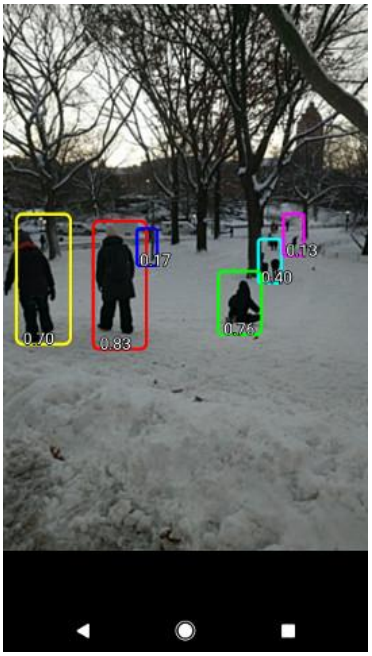
Το αποτέλεσμα μοιάζει με τις παρακάτω εικόνες. Ένα τετράγωνο πλαίσιο που αποτελεί τον «καμβα» (DrawView extends View) στο οποίο ο χρήστης ζωγραφίζει το ψηφίο, ένα πλήκτρο "DETECT" το οποίο συλλέγει τα pixels, τα μετατρέπει σε πίνακα αριθμών κινητής υποδιαστολής (float array) και τα εισάγει στον classifier και στην συνέχεια στο tensorflow μοντέλο. Η ετικέτα «resText» εμφανίζει το αποτέλεσμα και τις πιθανότητες αυτό να είναι σωστό (από τις μεθόδους getLabel() και getConf() αντίστοιχα). Τέλος, το πλήκτρο «CLEAR» αδειάζει όλα τα pixels του καμβά, για να ζωγραφίσει ο χρήστης νέο ψηφίο.



Εικόνα 35: Προβλέψεις MNIST σε Android συσκευή

Περισσότερα android demos

Στο προφίλ (repository) του Tensorflow στο GitHub διατίθενται μερικά demo applications για android συσκευές, που δίνουν μια καλύτερη ιδέα για τις δυνατότητες που προσφέρει το tensorflow όταν χρησιμοποιείται σε κινητές συσκευές. Μερικές από αυτές είναι οι εφαρμογές TF Classify, TF Detect και TF Speech, που κάνουν ταξινόμηση αντικειμένων από την κάμερα σε πραγματικό χρόνο, εντοπισμό αντικειμένων και οριοθέτηση στην εικόνα της κάμερας σε πραγματικό χρόνο, και αναγνώριση ηχητικών προτύπων (αναγνώριση λέξεων από ομιλία) αντίστοιχα.



Demo 1: TF Detect

Demo 2: TF Classify

Demo 3: TF Speech

5.2.2 IOS

Ο πιο αποτελεσματικός τρόπος να χρησιμοποιήσεις ένα εκπαιδευμένο μοντέλο μηχανικής μάθησης στο iOS πια είναι να πάρεις ένα προεκπαιδευμένο μοντέλο απο κάποιο εξωτερικό framework και να το μετατρέψεις σε ένα .mlmodel αρχείο το οποίο μπορεί να φορτωθεί σε ενα Xcode project και να χρησιμοποιηθεί μέσω της βιβλιοθήκης CoreML. Η Google και η Apple συνεργάστηκαν για να δημιουργήσουν ένα εργαλείο το οποίο μπορεί να μετατρέψει εκπαιδευμένα μοντέλα του Tensorflow σε .mlmodel αρχεία για να χρησιμοποιηθούν μέσω του CoreML και της Swift διευκολύνοντας έτσι τους προγραμματιστές πού δεν έχουν την ανάλογη εμπειρία και τεχνογνωσία για να μεταφέρουν τα μοντέλα αυτά μόνοι τους. Πριν από το CoreML για να χρησιμοποιήσει κάποιος ένα μοντέλο μηχανικής μάθησης στο iOS, ο μόνος τρόπος ήταν να το εκπαιδεύσεις μέσω των βιβλιοθηκών Metal και Accelerate οι οποίες είναι γραμμένες σε C++ και είναι δύσκολες στην χρήση τους, εκτός αυτού δεν είναι συμβατές με την Swift με αποτέλεσμα να χρειαζόταν wrapper interface μέσω της Objective-C για να χρησιμοποιηθούν απο ένα Swift developer.

Το εργαλείο που επιτρέπει την μετατροπή μοντέλων μηχανικής μάθησης σε mlmodel λέγεται coremltools και έχει την δυνατότητα να μετατρέψει μοντέλα σε mlmodel αρχεία απο τα παρακάτω frameworks:

- Scikit Learn
- LIBSVM
- Caffe
- Keras
- XGBoost

Για την μετατροπή από Tensorflow και MXNET σε mlmodel χρειάζονται δύο διαφορετικά εργαλεία tf-coreml και incubator-mxnet αντίστοιχα.

Εφόσον όλα αυτά τα εργαλεία είναι γραμμένα σε python μπορεί να χρησιμοποιηθεί το pip(package manager για βιβλιοθήκες γραμμενες σε python) με τον παρακάτω τρόπο για την εγκατάσταση του tf-coreml:

```
pip install -U tfcoreml
```

Επίσης μπορεί κάποιος να κάνει build απο το source με τον παρακάτω τρόπο:

```
git clone https://github.com/tf-coreml/tf-coreml.git
cd tf-coreml
```

Για την εγκατάσταση μέσω pip:

```
pip install -e .
```

η αλλιώς:

```
python setup.py bdist_wheel
```

Για να χρησιμοποιήσουμε τον converter χρειαζεται στο μοντέλο που εκπαιδεύουμε να έχουμε κώδικα που θα σώσει τον γράφο και το checkpoint που σταματησε η εκπαιδευση μαζί με όλα τα βάρη που δημιουργήθηκαν. Αυτή την λειτουργεία μας την παρέχει έτοιμη το Tensorflow μέσα σε μερικές γραμμές.

```
#Δημιουργια του Saver πριν ξεκινήσει η εκπαίδευση
saver = tf.train.Saver()

#Σωσιμό του session και του γράφου σε όποιο σημείο είμαστε ευχαριστημένη με το accuracy του μοντέλου
saver.save(session, "./model.ckpt")
```

```
tf.train.write_graph(session.graph_def, './', 'graph.pb')
```

Το επόμενο βήμα είναι να “παγώσουμε” το μοντέλο στην μορφή .pb γιατί τώρα το αρχείο graph.pb περιέχει μόνο τον γράφο χωρίς τα βάρη που υπολογίσαμε. Υπάρχει ένα python script που παρέχει η google το οποίο παραμετροποιήτε και μπορεί να παγώσει ένα μοντέλο. Το script λέγετε freeze_graph.py και παρέχε μαζί με το default installation του Tensorflow.

Αυτο που κάνει είναι να φορτώσει τον γράφο σαν αρχείο GraphDef(), να τραβήξει όλες της μεταβλητές του checkpoint αρχείου που έσωσε ο saver και στη συνέχεια να αντικαταστήσει κάθε Variable op με Const που περιέχει όλα τα αριθμητικά δεδομένα των βαρών που είναι αποθηκευμένα στα χαρακτηριστικά του. Στην συνέχεια αφαιρεί ολους τους κόμβους που είναι άχρηστοι για την εξαγωγή συμπερασμάτων και αποθηκεύει των τελικό γραφο GraphDef() σε ενα .pb αρχείο.

```
#Εντολή για την εξαγωγή ενός frozen graph, το softmax είναι το ονομα που δώθηκε στον τελικό κόμβο π.χ.( y_conv = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2, name="softmax") )
freeze_graph
--input_graph=/path/to/graph.pb
--input_checkpoint=/path/to/model.ckpt
--output_node_names=softmax
--output_graph=/path/to/frozen.pb
```

Τέλος μπορούμε να πάρουμε αυτο το παγωμένο γραφο και μέσψ του tfcoreml να εξάγουμε το .mlmodel αρχείο που χρειαζόμαστε.

```
#x ονομάσαμε το variable στο οποιο “ταίζουμε” της εικόνες, και το output ονομάστηκε softmax γιατι είναι το τελευταίο στρώμα
import tfcoreml
tfcoreml.convert(tf_model_path='frozen.pb',mlmodel_path='./deepMnist.mlmodel', output_feature_names=['softmax:0'], input_name_shape_dict = {'x:0': [1,784]})

#Log μετα την εκτέλεση του conversion
Shapes not found for 17 tensors. Executing graph to determine shapes.
1/35: Analysing op name: x ( type:  Placeholder )
Skipping name of placeholder
2/35: Analysing op name: Reshape/shape ( type:  Const )
3/35: Analysing op name: Reshape ( type:  Reshape )
4/35: Analysing op name: Variable ( type:  Const )
```

```
5/35: Analysing op name: Variable/read ( type:  Identity )
6/35: Analysing op name: Conv2D ( type:  Conv2D )
7/35: Analysing op name: Variable_1 ( type:  Const )
8/35: Analysing op name: Variable_1/read ( type:  Identity )
9/35: Analysing op name: add ( type:  Add )
10/35: Analysing op name: Relu ( type:  Relu )
11/35: Analysing op name: MaxPool ( type:  MaxPool )
12/35: Analysing op name: Variable_2 ( type:  Const )
13/35: Analysing op name: Variable_2/read ( type:  Identity )
14/35: Analysing op name: Conv2D_1 ( type:  Conv2D )
15/35: Analysing op name: Variable_3 ( type:  Const )
16/35: Analysing op name: Variable_3/read ( type:  Identity )
17/35: Analysing op name: add_1 ( type:  Add )
18/35: Analysing op name: Relu_1 ( type:  Relu )
19/35: Analysing op name: MaxPool_1 ( type:  MaxPool )
20/35: Analysing op name: Variable_4 ( type:  Const )
21/35: Analysing op name: Variable_4/read ( type:  Identity )
22/35: Analysing op name: Variable_5 ( type:  Const )
23/35: Analysing op name: Variable_5/read ( type:  Identity )
24/35: Analysing op name: Reshape_1/shape ( type:  Const )
25/35: Analysing op name: Reshape_1 ( type:  Reshape )
26/35: Analysing op name: MatMul ( type:  MatMul )
27/35: Analysing op name: add_2 ( type:  Add )
28/35: Analysing op name: Relu_2 ( type:  Relu )
29/35: Analysing op name: Variable_6 ( type:  Const )
30/35: Analysing op name: Variable_6/read ( type:  Identity )
31/35: Analysing op name: MatMul_1 ( type:  MatMul )
32/35: Analysing op name: Variable_7 ( type:  Const )
33/35: Analysing op name: Variable_7/read ( type:  Identity )
34/35: Analysing op name: add_3 ( type:  Add )
35/35: Analysing op name: softmax ( type:  Softmax )

 Core ML model generated. Saved at location: ./deepMnist.mlmodel

Core ML input(s):
 [name: "x__0"
type {
  multiArrayType {
    shape: 784
    dataType: DOUBLE
  }
```

```
}
]
Core ML output(s):
 [name: "softmax__0"
type {
  multiArrayType {
    shape: 10
    dataType: DOUBLE
  }
}
]
input {
  name: "x__0"
  type {
    multiArrayType {
      shape: 784
      dataType: DOUBLE
    }
  }
}
output {
  name: "softmax__0"
  type {
    multiArrayType {
      shape: 10
      dataType: DOUBLE
    }
  }
}
```

Ετσι έχουμε στην κατοχή μας ένα αρχείο το οποίο μπορεί να διαβαστεί από το Xcode και την CoreML βιβλιοθήκη.

5.2.3 ΔΗΜΙΟΥΡΓΙΑ DEMO iOS App

Η εφαρμογή είναι απλή, περιέχει δύο UI View στα οποια ο χρήστης μπορεί να ζωγραγίσει στο πρώτο με το χέρι του να πατήσει το κουμπί detect και στο αλλό UIView να εμφανιστεί ένα prediction.

Στο UIView στο οποίο ζωγραφίζει ο χρήστης χρησιμοποιούνται UIBezierPath για να ζωγραφίσουμε το σχήμα και ο controller του έχει μία μέθοδο που μετατρέπει και επιστρέφει το UIView σαν μία εικόνα τύπου CGContext.

```
func getViewContext() -> CGContext? {

        let colorSpace:CGColorSpace = CGColorSpaceCreateDeviceGray()
        let bitmapInfo = CGImageAlphaInfo.none.rawValue
        let context = CGContext(data: nil, width: 28, height: 28,
bitsPerComponent: 8, bytesPerRow: 28, space: colorSpace, bitmapInfo:
bitmapInfo)

        // 28x28 image
        context!.translateBy(x: 0 , y: 28)
        context!.scaleBy(x: 28/self.frame.size.width, y: -
28/self.frame.size.height)

        // put pixel data in context
        self.layer.render(in: context!)

        return context
}
```

Η αρχικοποίηση του μοντέλου είναι απλή αφου βάλουμε το αρχείο στο project μας.

```
import CoreML
let model = deepMnist()
```

Μετα μπορούμε να πάρουμε ένα prediction δίνοντας τα pixel μίας εικόνας στην μορφή ενός CVPixelBuffer

```
let context = drawView.getViewContext()
inputImage = context?.makeImage()
let pixelBuffer = UIImage(cgImage: inputImage).pixelBuffer()
let output = try? model.prediction(image: pixelBuffer!)
print(output?.classLabel)
```

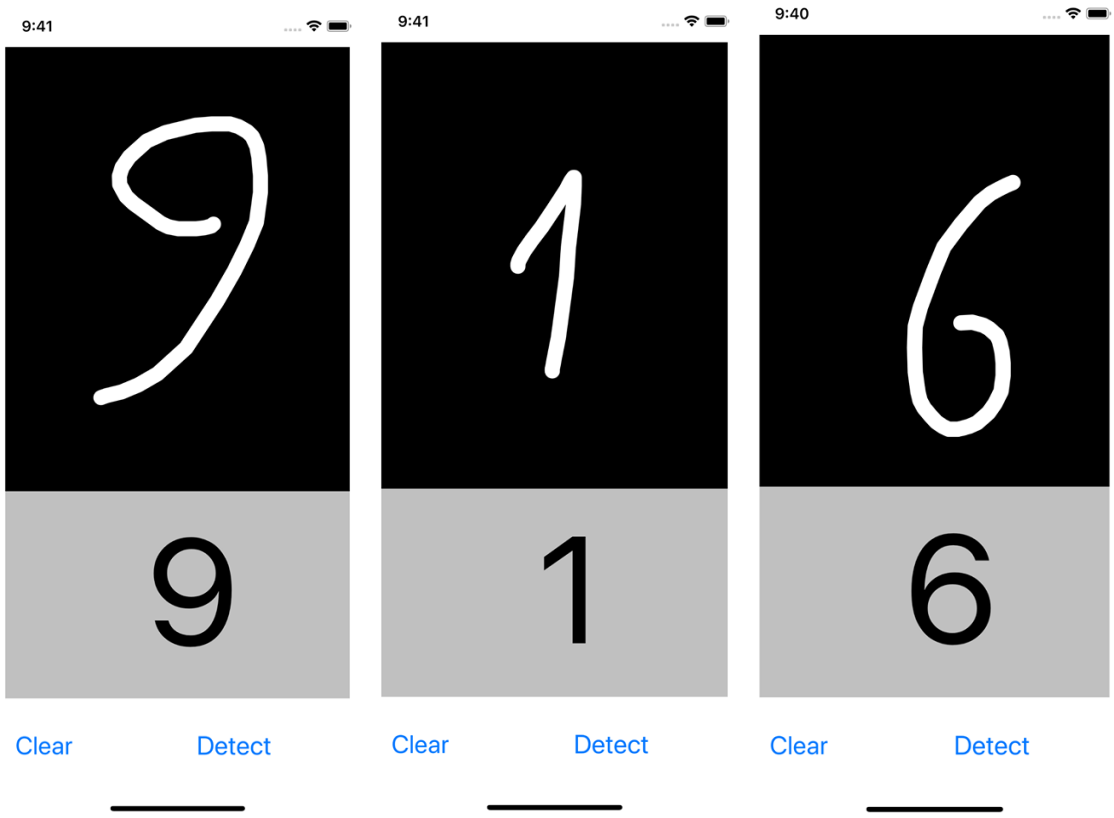
Η μέθοδος pixelBuffer υλοποιείται σαν extension του UIImage για ευκολία πρόσβασης από οπουδήποτε

```
extension UIImage {
```

```
    func pixelBuffer() -> CVPixelBuffer? {
        let width = self.size.width
        let height = self.size.height
        let attrs = [kCVPixelBufferCGImageCompatibilityKey: kCFBooleanTrue,
                     kCVPixelBufferCGBitmapContextCompatibilityKey:
kCFBooleanTrue] as CFDictionary
        var pixelBuffer: CVPixelBuffer?
        let status = CVPixelBufferCreate(kCFAllocatorDefault,
                                         Int(width),
                                         Int(height),
                                         kCVPixelFormatType_OneComponent8,
                                         attrs,
                                         &pixelBuffer)

        guard let resultPixelBuffer = pixelBuffer, status == kCVReturnSuccess
else {
            return nil
        }

        CVPixelBufferLockBaseAddress(resultPixelBuffer,
CVPixelBufferLockFlags(rawValue: 0))
        let pixelData = CVPixelBufferGetBaseAddress(resultPixelBuffer)

        let grayColorSpace = CGColorSpaceCreateDeviceGray()
        guard let context = CGContext(data: pixelData,
                                      width: Int(width),
                                      height: Int(height),
                                      bitsPerComponent: 8,
                                      bytesPerRow:
CVPixelBufferGetBytesPerRow(resultPixelBuffer),
                                      space: grayColorSpace,
                                      bitmapInfo:
CGImageAlphaInfo.none.rawValue) else {
                                        return nil
        }

        context.translateBy(x: 0, y: height)
        context.scaleBy(x: 1.0, y: -1.0)

        UIGraphicsPushContext(context)
        self.draw(in: CGRect(x: 0, y: 0, width: width, height: height))
```

```
        UIGraphicsPopContext()
        CVPixelBufferUnlockBaseAddress(resultPixelBuffer,
CVPixelBufferLockFlags(rawValue: 0))

        return resultPixelBuffer
    }
}
```

Το τελικό αποτέλεσμα μοιάζει με τις παρακάτω εικόνες. Στο μαύρο πλαίσιο ο χρήστης «ζωγραφίζει» ένα αριθμητικό ψηφίο και πατώντας το πλήκτρο “Detect” το μοντέλο αναγνωρίζει το ψηφίο και η εφαρμογή εμφανίζει στο γκρι πλαίσιο την πρόβλεψη:



Εικόνα 36: Προβλέψεις MNIST σε iOS συσκευή

ΑΝΑΚΕΦΑΛΑΙΩΣΗ

ΣΥΜΠΕΡΑΣΜΑΤΑ

Μέσα από την ενασχόλησή μας με τις τεχνολογίες του Machine Learning (ML), το Tensorflow και τις κινητές πλατφόρμες Android & iOS, καταλήγουμε στα ακόλουθα συμπεράσματα:

- Το ML προσφέρει πολλές δυνατότητες στους χρήστες και τους προγραμματιστές, και βοηθάει στην δημιουργία εφαρμογών και συστημάτων των οποίων η υλοποίηση μέχρι πριν μερικά χρόνια ήταν ανέφικτη.
- Το ML προσφέρει καινοτόμες λύσεις σε πολλές επιστήμες εκτός της πληροφορικής, και διευκολύνει την καθημερινότητα σε πολλούς τομείς όπως η υγεία, οικονομικές επιστήμες, marketing και πωλήσεις, κυβερνητικό έργο, αστικές (και όχι μόνο) συγκοινωνίες και πολλά ακόμα.
- Η εκπαίδευση δικτύων μηχανικής μάθησης είναι μια διαδικασία που εξαρτάται πάντα από το μέγεθος του δικτύου και των δεδομένων, όμως στις περισσότερες περιπτώσεις είναι χρονοβόρα και απαιτεί μηχανές μεγάλης επεξεργαστικής ισχύος.
- Παρ'όλα αυτά η χρήση των παραγόμενων μοντέλων βοηθάει την καθημερινότητά μας ιδιαίτερα μέσα από κινητές και ενσωματωμένες συσκευές, συχνά πολύ μικρότερης επεξεργαστικής ισχύος από της μηχανές που εκπαίδευσαν τα μοντέλα (GPU clusters, cloud κλπ).
- Υπάρχουν πολλά πλαίσια λογισμικού, έτοιμες λύσεις και κοινότητες που υποστηρίζουν την ανάπτυξη λογισμικών τεχνητής νοημοσύνης και μηχανικής μάθησης, καθώς οι απαιτήσεις αυξάνονται πολύ γρήγορα, και πλέον αφορούν και εμπορικούς σκοπούς.
- Το Tensorflow είναι ένα πλήρες –σε σχέση με τον ανταγωνισμό του– framework το οποίο παρέχει λύσεις εκπαίδευσης με πολλές επιλογές και παραλλαγές, όπως κατανεμημενη εκπαίδευση – με παραλληλισμό δεδομένων και παραλληλισμό μοντέλου, εκτέλεση σε κινητές (TensorflowMobile) και ενσωματωμένες συσκευές (TsLite).
- Το MNIST dataset είναι ένα μικρό και ιδανικό σύνολο δεδομένων για να γνωρίσει κανείς τον κόσμο της μηχανικής μάθησης, να εκπαιδεύσει κάποιο δίκτυο και να δοκιμάσει εφαρμογές της, καθώς λειτουργεί με εικόνες, και πολλές εφαρμογές μηχανικής μάθησης βασίζονται σε σύνολα δεδομένων εικόνων, ενώ παράλληλα είναι μικρό σε μέγεθος και εύκολα κατανοητό (μόνο ψηφία 0-9).

Η ΧΡΟΝΙΚΗ ΔΙΑΡΚΕΙΑ

Η χρονική διάρκεια από την έναρξη μέχρι την ολοκλήρωση της πτυχιακής εργασίας ήταν δυο ακαδημαϊκά εξάμηνα. Το πιο δύσκολο κομμάτι το εντοπίσαμε στην διαδικασία κατανόησης της λειτουργίας των συνελικτικών δικτύων, και στην συνέχεια την κατανόηση του Tensorflow. Από την ημερομηνία που ξεκινήσαμε την αναζήτηση πληροφοριών και ανάγνωση της τεκμηρίωσης του Tensroflow μέχρι και τις τελευταίες ημέρες παρατηρούσαμε πολλές αλλαγές στην τεκμηρίωση και

την λειτουργία του πλαισίου, αλλαγές όπως η διαχείριση των συνεδριών (tf.Sessions) μέχρι και την νέα έκδοση του πλαισίου για κινητές συσκευές (Tensorflow Lite ενώ εμείς ξεκινήσαμε με το Tensorflow for Mobiles). Το πιο χρονοβόρο κομμάτι, ήταν η διαδικασία συγγραφής και δοκιμών του κώδικα κατανεμημένης εκπαίδευσης (σε γλώσσα python) και η εκπαίδευση τελικά του μοντέλου με το MNIST dataset.

ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Εφόσον ο τομέας της μηχανικής μάθησης διαρκώς αναπτύσσεται και έχει τόσες προοπτικές ακόμα, είναι βέβαιο οτι υπάρχουν πολλές επιλογές για μελλοντικές επεκτάσεις των mobile εφαρμογών και της κατανεμημένης εκπαίδευσης. Ένα παράδειγμα επέκτασης των κινητών εφαρμογών (MNIST demo), είναι η αναγνώριση χειρόγραφων αριθμών από φωτογραφίες (ή live frames) που προέρχονται από την κάμερα της συσκευής και χρήση αυτών των ψηφιοποιημένων πλέον αριθμών (πχ αναγνώριση και κλήση ενός αριθμού τηλεφώνου από χειρόγραφη μορφή, μέσω της κάμερας). Όσο για την κατανεμημένη εκπαίδευση, ο αλγόριθμος μπορεί να βελτιστοποιηθεί και να λειτουργήσει σε ένα cluster με αρκετές δυνατές GPU, ώστε να εκπαιδεύσει παράλληλα μεγάλα μοντέλα και με μεγαλύτερα datasets από το MNIST.

ΕΜΠΕΙΡΙΕΣ

Η ανάθεση αυτού του θέματος για την πτυχιακή μας εργασία μας έδωσε μια ευκαιρία να γνωρίσουμε την μηχανική μάθηση και να κατανοήσουμε βασικές έννοιες, που μας ανοίγουν πολλούς δρόμους για το μέλλον, μιας και η μηχανική μάθηση εμπλέκεται με τόσους πολλούς τομείς και εφαρμογές. Είδαμε στην πράξη πως μετατρέπονται τα αναλογικά δεδομένα σε κατάλληλα ψηφιακά για να επεξεργαστούν από δίκτυα ΜΜ, γνωρίσαμε τι μπορεί να προσφέρει στην πράξη η μηχανική μάθηση και πως μπορεί αυτό να υλοποιηθεί. Επεκτείναμε τις γνώσεις μας επάνω στον προγραμματισμό εφαρμογών για κινητές συσκευές Android και iOS, ενώ παράλληλα πήραμε έναυσμα για περαιτέρω ενασχόληση με την μηχανική μάθηση στις κινητές συσκευές, γνωρίζοντας τις βασικές απαιτήσεις και έχοντας κάποιες ώρες ενασχόλησης με συγγραφή σχετικού κώδικα (και αποσφαλμάτωσης) στην εμπειρία μας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

ΚΑΤΕΡΙΝΑ ΓΕΩΡΓΟΥΛΗ: ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ, Μια εισαγωγική προσέγγιση.
https://repository.kallipos.gr/bitstream/11419/3382/1/02_chapter_04.pdf

ΚΩΝΣΤΑΝΤΙΝΟΣ ΔΙΑΜΑΝΤΑΡΑΣ: Μηχανική Μάθηση - Βαθιά Μάθηση και Εφαρμογές
http://cie.teiemt.gr/cie/wp-content/uploads/2015/05/%CE%9A%CE%B1%CE%B2%CE%AC%CE%BB%CE%B1-5-2-2017.pdf

Python - Machine Learning Tasks
https://pythonprogramminglanguage.com/machine-learning-tasks/

Difference Between Classification and Regression in Machine Learning
https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

ΚΩΝΣΤΑΝΤΙΝΟΣ ΔΙΑΜΑΝΤΑΡΑΣ – Μηχανική Μάθηση – Βασικές έννοιες
https://aetos.it.teithe.gr/~kdiamant/MachineLearning/MachineLearningLesson01.pdf

Ibrar Hussain - Clustering in Machine Learning
http://www.cad.zju.edu.cn/home/zhx/csmath/lib/exe/fetch.php?media=2011:presentation_ml_by_ibrar.pdf

Digital Trends - What is an artificial neural network
https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

Stanford University CS321n - Convolutional Neural Networks (CNNs / ConvNets)
http://cs231n.github.io/convolutional-networks/

Stanford University CS321n - Convolutional Neural Networks (CNNs / ConvNets)
http://cs231n.stanford.edu/

tensorflow.org
https://www.Tensorflow.org/programmers_guide/eager

Amy Unhuh - What is the TensorFlow machine intelligence platform
https://opensource.com/article/17/11/intro-Tensorflow

ESHRAT AILIOMANDI, MICHAEL J. FISCHER, NANCY A. LYNCH: Efficiency of Synchronous Versus Asynchronous Distributed Systems
https://groups.csail.mit.edu/tds/papers/Lynch/jacm83.pdf

tensorflow.org
https://www.Tensorflow.org/deploy/distributed

skymind.ai - Distributed Deep Learning, Part 1: An Introduction to Distributed Training of Neural Networks
https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

RICHA BHATIA - Top 10 Popular Publicly Available Datasets For Deep Learning Research
https://analyticsindiamag.com/top-10-popular-publicly-available-datasets-deep-learning-research/

Christos Christofidis - Awesome Deep Learning
https://github.com/ChristosChristofidis/awesome-deep-learning#datasets

tensorflow.org
https://www.Tensorflow.org/versions/r1.1/get_started/mnist/beginners

Yann Lecun - THE MNIST DATABASE of handwritten digits
http://yann.lecun.com/exdb/mnist/

Christopher Olah - Visualizing MNIST: An Exploration of Dimensionality Reduction
http://colah.github.io/posts/2014-10-Visualizing-MNIST/

Pranav Dar - 25 Open Datasets for Deep Learning Every Data Scientist Must Work With
https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets/

Andrei Frumusanu – HiSilicon Kirin 970 – Android SoC power & performance overview
https://www.anandtech.com/show/12195/hisilicon-kirin-970-power-performance-overview/5

Tensorflow.org
https://www.Tensorflow.org/mobile/tflite/demo_android

Dan Jarvis - Using a Pre-Trained TensorFlow Model on Android — Part 1
https://medium.com/capital-one-developers/using-a-pre-trained-Tensorflow-model-on-android-e747831a3d6

Tensorflow.org
https://www.Tensorflow.org/mobile/mobile_intro

AMIT SHEKHAR - Android-TensorFlow-Lite-Example
https://github.com/amitshekhariitbhu/Android-Tensorflow-Lite-Example

Alex Smola - What is the Parameter Server?
https://www.quora.com/What-is-the-Parameter-Server

Wikipedia.org
https://en.wikipedia.org/wiki/Blackboard_system

Quora - Shubhanshu Mishra, PhD Information Science - How is PyTorch different from TensorFlow?
https://www.quora.com/How-is-PyTorch-different-from-Tensorflow-What-are-the-advantages-of-using-one-vs-the-other-When-should-I-use-one-or-the-other

Apple's CoreML - TensorFlow to CoreML Converter
https://github.com/tf-coreml/tf-coreml

Apple
https://github.com/apple/coremltools

Jeevan Biswas - What Is Machine Learning As A Service (MLaaS)?
https://analyticsindiamag.com/what-is-machine-learning-as-a-service-mlaas/

Deep learning 4j - A Beginner's Guide to Deep Convolutional Neural Networks (CNNs)
https://deeplearning4j.org/convolutionalnetwork

Tim Dettmers - How to Parallelize Deep Learning on GPUs Part 1/2: Data Parallelism
http://timdettmers.com/2014/10/09/deep-learning-data-parallelism/

Tim Dettmers - How to Parallelize Deep Learning on GPUs Part 2/2: Model Parallelism
http://timdettmers.com/2014/11/09/model-parallelism-deep-learning/

Jim Dowling - Distributed TensorFlow
https://www.oreilly.com/ideas/distributed-tensorflow