ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ &ΕΠΙΚΟΙΝΩΝΙΩΝ

«ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΜΟΝΤΕΛΩΝ ΣΥΝΕΛΙΚΤΙΚΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΓΙΑ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ ΕΙΚΟΝΩΝ»

«COMPARISON OF CONVOLUTIONAL NEURAL NETWORKS MODELS FOR IMAGE CLASSIFICATION»

Πτυχιακή Εργασία του

Ίγκορ Σπιριντόνοβ (4116)

Επιβλέπων: Δρ. Αλκιβιάδης Τσιμπίρης

Σέρρες, Φεβρουάριος 2020

Υπεύθυνη Δήλωση : Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Πληροφορικής &Επικοινωνιών του Τ.Ε.Ι. Σερρών.

ΠΕΡΙΛΗΨΗ

Ο σκοπός της συγκεκριμένης πτυχιακής εργασίας είναι η σύγκριση αρχιτεκτονικών νευρωνικών δικτύων για κατηγοριοποίηση εικόνων. Πιο συγκεκριμένα η πτυχιακή εργασία απαρτίζεται από 2 μέρη. Το πρώτο μέρος αφορά το θεωρητικό κομμάτι της Μηχανικής Μάθησης. Στο πρώτο κεφάλαιο παρουσιάζονταιιστορικά στοιχεία για την Μηχανική Μάθηση και εξηγείται η λειτουργία των συνελικτικών νευρωνικών δικτύων καθώς και τα δομικά τους στοιχεία. Επίσης αναφέρονται τα πιο εδραιωμένα συνελικτικά νευρωνικά δίκτυα και τα πεδία εφαρμογής τους. Το δεύτερο μέρος της πτυχιακής εργασίας περιλαμβάνει την υλοποίηση του προγράμματος. Στα πλαίσια της πτυχιακής εργασίας δημιουργήθηκε μια βιβλιοθήκη η οποία περιλαμβάνει μεθόδους για την δημιουργία συνελικτικών δικτύων χωρίς την χρήση εξωτερικών βιβλιοθηκών. Επίσης στο δεύτερο μέρος υπάρχει το κομμάτι της δημιουργίας, εκπαίδευσης και σύγκρισης των συνελικτικών νευρωνικών δικτύων. Τα μοντέλα που εκπαιδεύτηκαν είναι το LeNet-5 που εκπαιδεύτηκε με την χρήση της βιβλιοθήκης που αναπτύχθηκε στα πλαίσια της πτυχιακής εργασίας, LeNet-5 εκπαιδευμένο με την βιβλιοθήκη Keras καθώς και μία παραλλαγή του μοντέλου VGG. Τέλος παρουσιάζονται τα αποτελέσματα της σύγκρισης των μοντέλων.

ABSTRACT

The purpose of this thesis is to compare models of convolutional neural networks for image classification. In particular, the bachelor thesis consists of two chapters. The first chapter includes the theoretical basis of Machine Learning and the history of machine learning. The building blocks of convolutional neural networks are also presented in the first chapter. It is also essential to study the most widely used models and the domains they are used. The second part of the bachelor thesis consists of the implementation of the library with utilities to create and train the models without using third party libraries. The second part includes the part of the creation and the training of models using the developed library and the comparison of the neural network architectures. The models that were developed for this thesis are the LeNet-5 that was trained using the developed library, the LeNet-5 that was trained using the Keras library, and a variation of the VGG model. At last, the results of the comparison are presented.

ΕΥΧΑΡΙΣΤΙΕΣ

Η συγγραφή του παρόντος συγγράμματος έγινε με την υποστήριξη και συμπαράσταση πολλών ανθρώπων του οποίους θα ήθελα να ευχαριστήσω θερμά. Αρχικά ευχαριστώ τον επιβλέποντα καθηγητή Δρ. Αλκιβιάδη Τσιμπίρη που μου έδωσε την δυνατότητα να ασχοληθώ με το συγκεκριμένο θέμα καθώς και για την άμεση ανταπόκριση του σε κάθε ζήτημα που προέκυπτε κατά την διάρκεια της εργασίας και για την πολύτιμη καθοδήγηση που μου παρείχε.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου Αλεξάντερ Μπαντρισβίλι και Δημήτριο Σουραϊλίδη που με στήριξαν εμπράκτως, είτε άμεσα είτε έμμεσα, στην προσπάθεια περάτωσης αυτής της εργασίας, αλλά και γενικότερα για την ολοκλήρωση των σπουδών μου και την απόκτηση του πτυχίου μου.

Τέλος θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματος που μου μετέφεραν την γνώση τους κατά τη διάρκεια των σπουδών μου στο τμήμα Μηχανικών Πληροφορικής.

Πίνακας Περιεχομένων

1 Εισαγωγή

1.1 Τεχνητή Νοημοσύνη

Ο όρος της τεχνητή νοημοσύνης (ΤΝ) (Artificial Intelligence AI) σχετίζεται με τον κλάδο της πληροφορικής που έχει να κάνει με την σχεδίαση υπολογιστικών συστημάτων τα οποία μιμούνται την ανθρώπινη συμπεριφορά υποδηλώνοντας την παρουσία στοιχειώδους ευφυΐας, καθώς και την υλοποίησή τους. Παραδείγματα αποτελούν η εκμάθηση και προσαρμογή στο περιβάλλον, η εξαγωγή συμπερασμάτων και γενικότερα η επίλυση απλών ή και πολύπλοκων προβλημάτων.

Μεγάλος αριθμός επιστημών συναντάται και συνεισφέρει στην τεχνητή νοημοσύνη, όπως για παράδειγμα η επιστήμη της πληροφορικής, της ψυχολογίας και της φιλοσοφίας, η νευρολογία, η επιστήμη των μηχανών, ακόμη και η επιστήμη της γλωσσολογίας προκειμένου να γίνει εφικτή η σύνθεση ευφυούς συμπεριφοράς, η εκμάθηση και προσαρμογή στο εκάστοτε περιβάλλον μηχανών και υπολογιστών συγκεκριμένου συνήθως σκοπού. Η ΤΝ διαχωρίζεται σε δυο κομμάτια, την συμβολική και την υπο-συμβολική νοημοσύνη. Η πρώτη συνίσταται στην προσπάθεια εξομοίωσης της ανθρώπινης συμπεριφοράς με χρήση ειδικών αλγορίθμων, μέσα από ένα σύνολο συμβόλων και λογικών κανόνων υψηλού επιπέδου ενώ η δεύτερη στόχο έχει την προσέγγιση ή ακόμα και την αναπαραγωγή της ανθρώπινης ευφυΐας μέσα από στοιχειώδη αριθμητικά μοντέλα τα οποία επαγωγικά συνθέτουν συμπεριφορές που υποδηλώνουν ευφυΐα, προσεγγίζοντας πραγματικές βιολογικές συμπεριφορές όπως η διαδικασία της εξέλιξης και η λειτουργία του ανθρώπινου εγκεφάλου.

Σε σχέση με κάποιο επιθυμητό στόχο η ΤΝ μπορεί να χωριστεί σε ένα ευρύτερο σύνολο τομέων, παραδείγματος χάριν στην μηχανική εκμάθηση, την επίλυση προβλημάτων, τα συστήματα γνώσης κλπ. Σε μεγάλο αριθμό περιπτώσεων υπάρχει επικάλυψη με συναφή επιστημονικά πεδία, πχ υπολογιστική όραση, σύνθεση και αναγνώριση φυσικής γλώσσας και ρομποτική, οι οποίες μπορούν να θεωρηθούν ανεξάρτητες συνιστώσες-πεδία της σύγχρονης ΤΝ.

Κινηματογραφικές ταινίες με θέμα επιστημονικής φαντασίας αλλά και λογοτεχνίας από τις αρχές της δεκαετίας 1920 έχουν αναδείξει με την ΤΝ δημιουργώντας αξιόλογα έργα και οπτικοποιώντας μια ιδεατή πραγματικότητα όπου ρομπότ και [14] υπολογιστικά συστήματα συνυπάρχουν με τους ανθρώπους και τους εξυπηρετούν στην καθημερινότητα. Ωστόσο, η λανθασμένη εντύπωση που έχει προκληθεί στο ευρύ κοινό περί κατασκευής μηχανικών ανδροειδών, αυτοσυνείδητων υπολογιστικών συστημάτων με σκοπό ακόμη και την αντικατάσταση του ανθρώπου δεν σπανίζουν, έχοντας επηρεάσει ακόμα και τους πρώτους επιστήμονες σχετικούς με τον τομέα. Στην πραγματικότητα, οι ερευνητές του τομέα της Τεχνίτης Νοημοσύνης έχουν ως στόχο την κατασκευή λογισμικού και μηχανικών συστημάτων ικανών να επιλύουνπροβλήματα πραγματικού περιεχομένου διαφόρων τύπων (ασθενής ΤΝ), ενώ αρκετοί αποβλέπουν στην προσομοίωση της πραγματικής ευφυΐας, την λεγόμενη ισχυρή ΤΝ.

Η ΤΝ την εποχή μας,αποτελεί ένα από τα ταχύτερα εξελισσόμενα πεδία της επιστήμης, ενώ με την χρήση εργαλείων των εφαρμοσμένων μαθηματικών και επιστήμες μηχανικών έχει ξεφύγει από τα πλαίσια της θεωρητικής πληροφορικής. Τέλος, μελετάται από την ηλεκτρονική μηχανική ενώ αποτελεί θεμέλιο του διεπιστημονικού πεδίου της γνωσιακής επιστήμης.

1.2 Μηχανική Μάθηση - Γενικά

Η μηχανική μάθηση (machine learning), αποτελεί ένα από τα σημαντικότερα κομμάτια της ΤΝ, η οποία αφορά την ανάπτυξη αλγορίθμων κατάλληλων που θα δώσουν τη δυνατότητα της “εκμάθησης” στους υπολογιστές. Το λογισμικό που χρησιμοποιείται από τους υπολογιστές γίνεται πλέον ευέλικτο και προσαρμόσιμο με βάση την ανάλυση των δεδομένων που λαμβάνουν, αντί της κλασικής πλέον προσαρμογής τους με βάση την διαίσθηση του μηχανικού που προγραμματίζει κάποιο σύστημα. Η ουσία της μηχανικής μάθησης συνοψίζεται στη χρήση αλγορίθμων, ικανών να αναγνωρίζουν μοτίβα στα δεδομένα προκειμένου να λάβουν αποφάσεις, βασιζόμενες στην στατιστική, τη θεωρία των πιθανοτήτων και την βελτιστοποίηση. Χάρις την Μηχανική Μάθηση, απολαμβάνουμε υπηρεσίες όπως φίλτρα ανεπιθύμητης αλληλογραφίας, αναγνώριση κειμένου και φωνής, αξιόπιστες μηχανές αναζήτησης στο διαδίκτυο, προκλητικούς αντιπάλους στο σκάκι και σύντομα σε αυτό-οδηγούμενα μέσα μεταφοράς.

Ανάλογα με το επιθυμητό αποτέλεσμα, οι αλγόριθμοι ΤΝ χωρίζονται στις εξής κατηγορίες:

• **Επιτηρούμενη μάθηση ή μάθηση υπο επίβλεψη (supervised learning)**, όπου ο αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδουςδείγματα (labeled examples) σε γνωστές-επιθυμητές εξόδους (σύνολο εκπαίδευσης), κάνοντας προβλέψεις και διορθώνοντας τις προβλέψεις σε [15] περίπτωση λάθους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής για εισόδους με άγνωστη έξοδο (σύνολο ελέγχου).

• **Μη επιτηρούμενη μάθηση ή μάθηση χωρίς επίβλεψη (unsupervised learning),** όπου ο αλγόριθμος κατασκευάζει ένα μοντέλο για κάποιο σύνολο εισόδων χωρίς να γνωρίζει επιθυμητές εξόδους-δείγματα (unlabeled examples) για το σύνολο εκπαίδευσης ανακαλύπτοντας δομές, όπως για παράδειγμα εξάγοντας γενικούς κανόνες.

• **Ημι-επιτηρούμενη μάθηση (semi-supervised learning),** όπου τα δεδομένα εκπαίδευσης είναι μια μίξη γνωστών και αγνώστων δειγμάτων (mixture of labeles & unlabeled examples), όπου υπάρχει ένα επιθυμητό πρόβλημα πρόβλεψης, αλλά το μοντέλο πρέπει να μάθει δομές για να οργανώσει τα δεδομένα και να κάνει προβλέψεις.

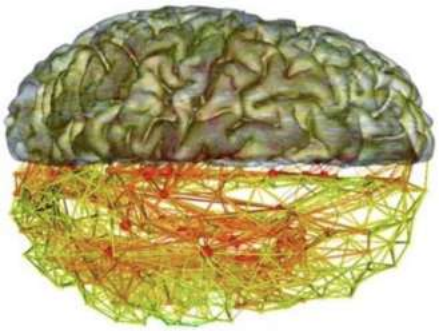
1.3 Νευρωνικά Δίκτυα - Γενικά

Τα Νευρωνικά Δίκτυα ή ΝΔ (Neural Networks ή NN), χωρίζονται σε Βιολογικά Νευρωνικά Δίκτυα (ΒΝΔ) ή Biological Neural Networks (BNN) και σε Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) ή Artificial Neural Networks (ANN). Τα ΒΝΔ αποτελούν μέρος του κεντρικού νευρικού συστήματος βιολογικών συστημάτων, παραδείγματος χάριν του ανθρώπου, ενώ συνίστανται από βιολογικό ιστό, χημικές ουσίες και ηλεκτρικά σήματα τα οποία τα διαχωρίζουν από τα ΤΝΔ που προσπαθούν να μιμηθούν τα πρώτα μέσα από ένα σύνολο ηλεκτρονικών και μηχανικών συστημάτων συνοδευόμενα από ευφυείς αλγορίθμους.

1.3.1 Ανθρώπινος Εγκέφαλος – Βιολογικοί Νευρώνες

Η ιδέα ότι ο ανθρώπινος εγκέφαλος είναι ένας υπολογιστής, έχει φέρει στο προσκήνιο της γνωσιακής επιστήμης μια σειρά από θέματα που έχουν να κάνουν με την εξελικτική θεωρία και φυσικά την εξέλιξη του εγκεφάλου. Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ), ξεκίνησαν ως μία προσπάθεια μοντελοποίησης της συμπεριφοράς του ανθρώπινου εγκεφάλου, ωστόσο σήμερα η εξέλιξη τους είναι σχεδόν ανεξάρτητη, παρόλα αυτά, ομοιότητες μπορούν ακόμα να εντοπιστούν.

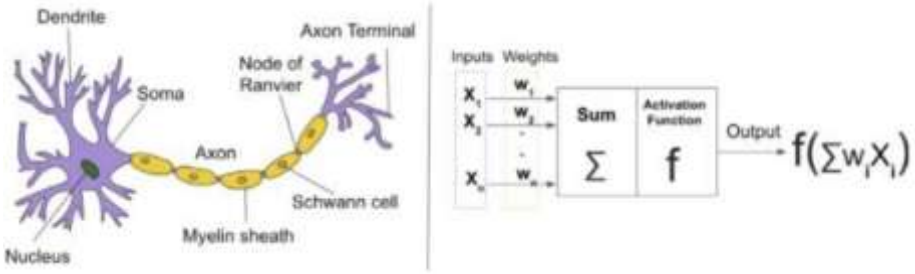
Οι βιολογικοί νευρώνες, όπως απεικονίζονται στο Σχήμα 1, αποτελούνται από τρία βασικά τμήματα που είναι το σώμα, ο άξονας και οι δενδρίτες. Αναλυτικότερα οι δενδρίτες, λαμβάνουν σήματα από γειτονικούς νευρώνες, τα σήματα αυτά είναι ηλεκτρικοί παλμοί που διαδίδονται μεταξύ του άξονα του νευρώνα πομπού και των δενδριτών του νευρώνα δέκτη με την βοήθεια χημικών διεργασιών. Το σημείο των χημικών διεργασιών, όπου ο άξονας ενός νευρώνα μεταδίδει το σήμα στους δενδρίτες του επόμενου λέγεται σύναψη. Αναφορικά αυτές οι διεργασίες μεταβάλλουν τα εισερχόμενα σήματα αλλάζοντας τη συχνότητα τους. Στην συνέχεια το σώμα αθροίζει τα εισερχόμενα σήματα και όταν αρκετά σήματα έχουν ληφθεί αποστέλλει το επεξεργασμένο σήμα στους γειτονικούς του νευρώνες μέσω του άξονα και η διαδικασία ξεκινά ξανά. Έτσι κάθε νευρώνας δέχεται πολλά σήματα ως είσοδο και μετά την επεξεργασία τους διαδίδει μόνο ένα σε όλους τους νευρώνες με τους οποίους συνδέεται. (Πλεύρου, 2012)



Σχήμα 1: Απεικόνιση Βιολογικών Νευρωνικών Δικτύων

Σε βιολογικά νευρωνικά δίκτυα όπως ο ανθρώπινος εγκέφαλος, η εκμάθηση επιτυγχάνεται με την πραγματοποίηση μικρών τροποποιήσεων σε μια υπάρχουσα αναπαράσταση, η διαμόρφωσή της οποίας περιέχει σημαντικές πληροφορίες. Τα πλεονεκτήματα των συνδέσεων μεταξύ των νευρώνων ή των βαρών 16 δεν ξεκινούν ως τυχαία, ούτε και η δομή των συνδέσεων, όπως συμβαίνει συνήθως στα τεχνητά νευρωνικά δίκτυα. Αυτή η αρχική κατάσταση είναι εν μέρει γενετική και είναι το υποπροϊόν της εξέλιξης. Με την πάροδο του χρόνου, το δίκτυο μαθαίνει πώς να εκτελεί νέες λειτουργίες προσαρμόζοντας τόσο την τοπολογία όσο και το βάρος των συνδέσεων των νευρώνων.

Τα Βιολογικά Νευρωνικά Δίκτυα σε αντίθεση με τα Τεχνητά, συνήθως εκπαιδεύονται από το μηδέν, χρησιμοποιώντας μια σταθερή διάταξη που επιλέγεται για το συγκεκριμένο πρόβλημα. Προς το παρόν, οι διατάξεις τους δεν αλλάζουν με την πάροδο του χρόνου και τα βάρη αρχικοποιούνται τυχαία και παραμετροποιούνται μέσω ενός αλγορίθμου βελτιστοποίησης για να χαρτογραφούν τις συνθέσεις των ερεθισμάτων εισόδου σε μια επιθυμητή συνάρτηση εξόδου. Ωστόσο, τα ΤΝΔ μπορούν επίσης να μάθουν με βάση μια προϋπάρχουσα κατάσταση. Αυτή η διαδικασία συνίσταται στην προσαρμογή των βαρών από μια προ-εκπαιδευμένη τοπολογία δικτύου σε ένα σχετικά αργό ρυθμό εκμάθησης για να αποδίδει καλά στα πρόσφατα παρεχόμενα δεδομένα

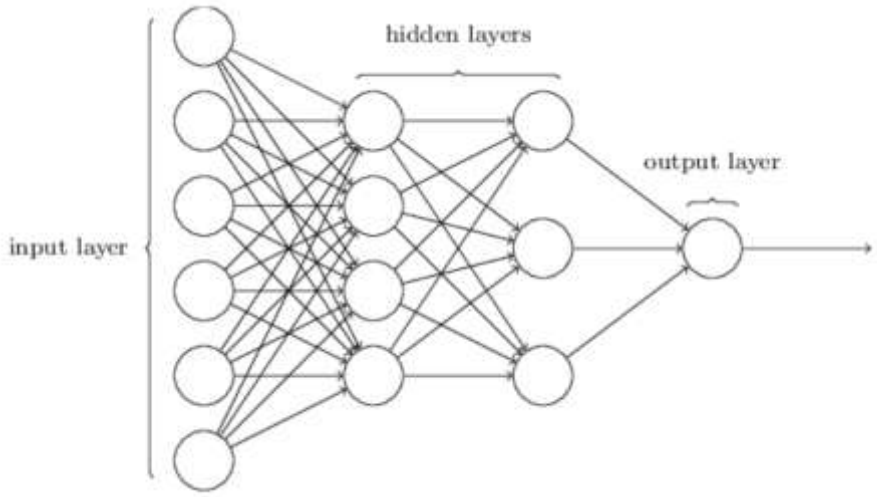


κατάρτισης εισόδου.

Σχήμα 2: Αριστερά: Απεικόνιση Βιολογικού Νευρωνικού Δικτύου, Δεξιά: Απεικόνιση Τεχνητού Νευρωνικού Δικτύου. Από την Wikipedia

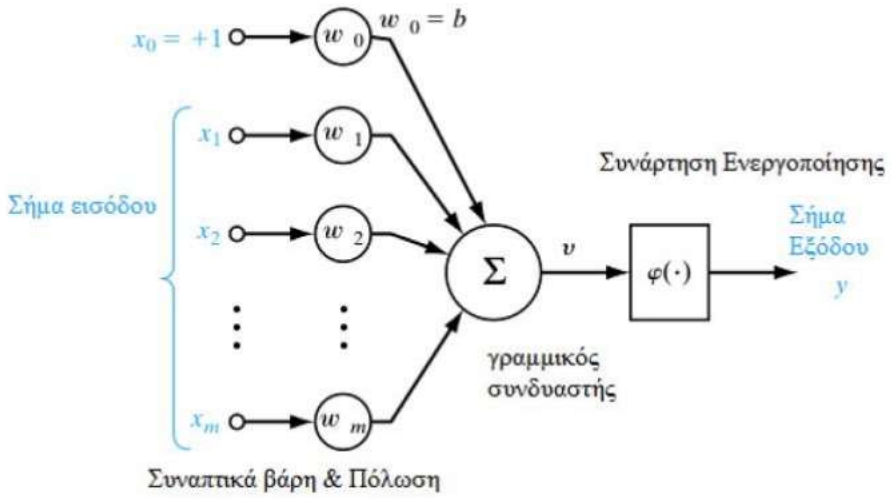
1.3.2 Το μοντέλο του Τεχνητού Νευρώνα

Το 1943 οι Warren McCulloch και Walter Pitts, δημιούργησαν ένα υπολογιστικό μοντέλο για νευρωνικά δίκτυα, βασισμένο σε μαθηματικά και αλγορίθμους και το ονόμασαν λογική κατωφλιού (threshold logic). Η κατάσταση ενός τεχνητού νευρώνα περιγράφεται από έναν δυαδικό αριθμό □, όπου όταν □= 0, ο νευρώνας είναι αδρανής και αντίστοιχα όταν □= 1, ο νευρώνας ενεργοποιείται. Η βασική δομή του, φαίνεται στο σχήμα 3. (Nielsen, 2015)



Σχήμα 3: Το μοντέλο ενός τεχνητού νευρώνα.

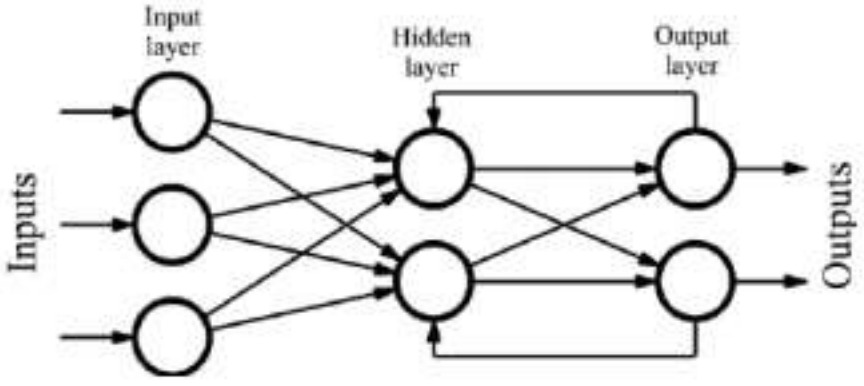
Συνδυάζοντας πολλαπλούς νευρώνες μαζί κατασκευάζεται ένα νευρωνικό δίκτυο. Ένα νευρωνικό δίκτυο, αποτελείται από κόμβους με διασυνδεδεμένες συναπτικές συνδέσεις και συναρτήσεις ενεργοποίησης. Υπάρχουν τρεις τύποι νευρώνων: οι νευρώνες εισόδου, οι νευρώνες εξόδου και οι υπολογιστικοί νευρώνες ή κρυμμένοι νευρώνες, οι οποίοι βρίσκονται στο στρώμα εισόδου, το στρώμα εξόδου και το κρυμμένο στρώμα αντίστοιχα (hidden layer). Ο όρος κρυμμένο, αναφέρεται στο ότι το στρώμα αυτό δεν είναι άμεσα ορατό από τα επίπεδα εισόδου και εξόδου. Τα σήματα εξόδου από ένα επίπεδο, χρησιμοποιούνται ως σήματα εισόδου για το επόμενο επίπεδο. Υπάρχουν νευρωνικά δίκτυα με περισσότερα από ένα κρυμμένο στρώμα, με τα οποία θα ασχοληθούμε και εκτενέστερα στην πορεία της παρούσας εργασίας. Ένα τέτοιο δίκτυο, παρουσιάζεται στο σχήμα 4, το οποίο ονομάζεται πλήρως συνδεδεμένο, με την έννοια ότι κάθε κόμβος ενός επιπέδου συνδέεται με κάθε άλλο κόμβο του επόμενου επιπέδου.



Σχήμα 4: Ένα πλήρες συνδεδεμένο νευρωνικό δίκτυο εμπρόσθιας τροφοδότησης

Τα ΤΝΔ κατηγοριοποιούνται ανάλογα με την αρχιτεκτονική τους και τον τρόπο με τον οποίο συνδέονται οι νευρώνες μεταξύ τους. Τα πιο συνηθισμένα δίκτυα ονομάζονται feedforward νευρωνικά δίκτυα, το οποίο σημαίνει ότι η πληροφορία εντός του δικτύου είναι πάντα προς τα εμπρός και δεν επιτρέπεται η προς τα πίσω τροφοδότηση του δικτύου.

Ωστόσο, υπάρχουν άλλα είδη νευρωνικών δικτύων, στα οποία επιτρέπεται η προς τα πίσω τροφοδότηση, τα οποία ονομάζονται Recurrent Neural Networks (RNN). Με τα RNN θα ασχοληθούμε εκτενέστερα στην πορεία της εργασίας αυτής και κυρίως με ένα συγκεκριμένος είδος αυτών, τα Long Short – Term Memory (LSTM) δίκτυα. Σε αυτά τα δίκτυα οι αλγόριθμοι μάθησης είναι λιγότερο ισχυροί, παρόλα αυτά είναι πολύ πιο κοντά στα βιολογικά νευρωνικά δίκτυα από τα feedforward και μπορούν να δώσουν λύσεις σε ορισμένα προβλήματα στα οποία τα πρώτα δυσκολεύονται.



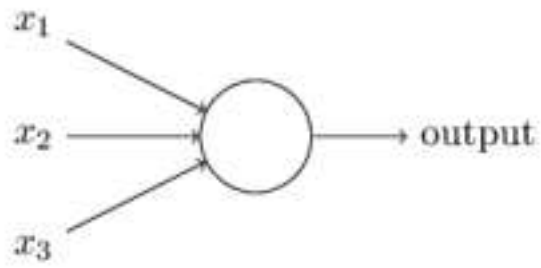
Σχήμα 5:Recurrent Neural Network By Özel, T. and Davim J.P. (2009) Modeling and optimization of the machining processes and systems

1.4 Συναρτήσεις Ενεργοποίησης

Αυτές οι συναρτήσεις έχουν χρησιμότητα όταν θέλουμε να προσδιορίσουμε την τιμή εξόδου (output), ενός ΝΔ. Αυτές χωρίζονται σε δύο ειδών συναρτήσεις, στις γραμμικές και στις μη-γραμμικές συναρτήσεις ενεργοποίησης. Στη συνέχεια θα παρουσιάσουμε μερικές από τις πιο βασικές μη-γραμμικές συναρτήσεις ενεργοποίησης, καθώς είναι και αυτές που χρησιμοποιούνται στην πράξη, αφού οι γραμμικές δεν βοηθάνε με την πολυπλοκότητα των συνηθισμένων δεδομένων που τροφοδοτούνται τα νευρωνικά δίκτυα.

1.4.1Συνάρτηση Κατωφλιού - Perceptrons

Τα Perceptrons αναπτύχθηκαν τις δεκαετίες του 50’ και του 60’ από τον επιστήμονα Frank Rosenblatt. Σήμερα κυρίως χρησιμοποιούνται άλλα μοντέλα ΤΝΔ, το κυριότερο εκ των οποίων είναι ο Σιγμοειδές Νεύρωνας. Θα ξεκινήσουμε τη αναφορά μας όμως από τα perceptrons. Ένα perceptron δέχεται πολλές δυαδικές εισροές □1, □2, … , □□και παράγει ένα μοναδικό δυαδικό output.



Σχήμα 6:Απεικόνιση εισροών σε ένα Perceptron By Michael Nielsen - Neural Networks and Deep Learning

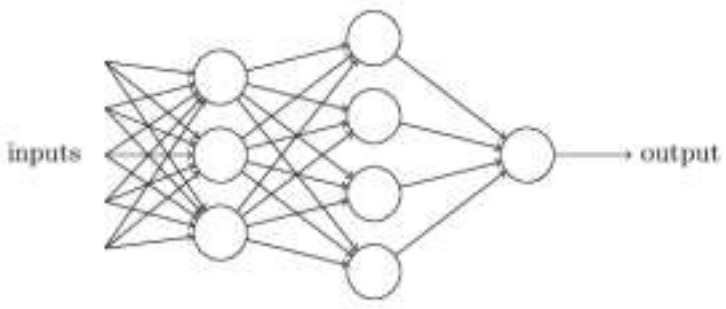
Ο Rosenblatt πρότεινε έναν απλό κανόνα για τον υπολογισμό του output. Εισήγαγε τα βάρη, □1, □2, … , □□, τα οποία είναι πραγματικοί αριθμοί που εκφράζουν τη σημασία των αντίστοιχων εισροών στο output. Το output του νευρώνα, 0 ή 1,

καθορίζεται από το εάν το σταθμισμένο άθροισμα ∑□□□□□είναι μικρότερο ή μεγαλύτερο από κάποια τιμή κατωφλίου (threshold). Ακριβώς όπως τα βάρη, το όριο είναι ένας πραγματικός αριθμός που είναι μια παράμετρος του νευρώνα. Δηλαδή με πιο ακριβείς αλγεβρικούς όρους:

$$output = \varphi(v) = \begin{cases} 0 \; if \; v = \sum_j w_j x_j \leq threshold \\ 1 \; if \; v = \sum_j w_j x_j > threshold \end{cases}$$

Το perceptron λειτουργεί ακριβώς με το παραπάνω απλό μαθηματικό μοντέλο, δηλαδή βάσει της βαρύτητας κάθε στοιχείου που έχει διαθέσιμου και ανάλογα την τιμή του κατωφλιού που έχουμε ορίσει λαμβάνει τις αποφάσεις

Προφανώς, το perceptron δεν είναι ένα πλήρες μοντέλο ανθρώπινης λήψης αποφάσεων. Αλλά είναι ένα μοντέλο το οποίο μπορεί να ζυγίσει διάφορα είδη αποδεικτικών στοιχείων για να πάρει αποφάσεις. Και θα πρέπει να φαίνεται εύλογο ότι ένα πολύπλοκο δίκτυο perceptrons θα μπορούσε να λάβει πολύ πιο λεπτές αποφάσεις.



Σχήμα 7:Πολύπλοκο Δίκτυο από Perceptrons By Michael Nielsen - Neural Networks and Deep Learning

Στο παραπάνω σχήμα, η πρώτη στήλη από perceptrons, την οποία ονομάζουμε ως πρώτο στρώμα (layer) από perceptrons, λαμβάνει τρεις απλές αποφάσεις σταθμίζοντας τα αποδεικτικά στοιχεία που του έχουν δοθεί. Αντίστοιχα τα perceptrons του δεύτερου στρώματος, λαμβάνουν αποφάσεις σταθμίζοντας τα αποτελέσματα του πρώτου στρώματος, ώστε ένα perceptron του δεύτερου στρώματος, να μπορεί να πάρει μία απόφαση σε ένα πιο πολύπλοκο επίπεδο από τον πρώτο στρώμα και ακολούθως ακόμα πιο πολύπλοκες αποφάσεις μπορούν να παρθούν από το τρίτο στρώμα. Έτσι με αυτό τον

τρόπο ένα TNN με πολλαπλά στρώματα μπορεί να λάβει αποφάσεις με πολύ πιο αποτελεσματικό τρόπο.

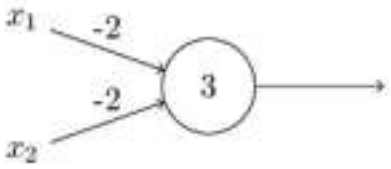
Μπορούμε να απλοποιήσουμε τον τρόπο που περιγράφουμε τα perceptrons. Η προϋπόθεση ∑□□□□□>□ℎ□□□ℎ□□□, δεν είναι πολύ εύχρηστη, γι' αυτό μπορούμε να κάνουμε δύο αλλαγές για να το απλοποιήσουμε. Αρχικά μπορούμε να θεωρήσουμε το ∑□□□□□ως □· □, όπου □και □είναι διανύσματα των οποίων οι συνιστώσες είναι τα βάρη και οι είσοδοι αντίστοιχα.

Έπειτα μπορούμε να μετακινήσουμε το κατώφλι στην άλλη πλευρά της ανίσωσης και να το αντικαταστήσουμε με το perceptron's bias(1) , □≡ − □ℎ□□□ℎ□□□, οπότε τώρα θα προκύπτει ότι

$$output = \begin{cases} 0 \; if \; \boldsymbol{w} \cdot \boldsymbol{x} + b \leq 0 \\ 1 \; if \; \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \end{cases}$$

Τη πόλωση (bias) μπορούμε να τη θεωρήσουμε ως ένα μέτρο του πόσο εύκολου είναι να παράξει το perceptron ως έξοδο την τιμή 1. Για ένα perceptron με μια πολύ μεγάλη πόλωση, είναι εξαιρετικά εύκολο να βγάλει ένα ως έξοδο την τιμή 1. Αλλά αν η πόλωση είναι πολύ αρνητική, τότε είναι δύσκολο για το perceptron να βγάλει τιμή 1.

Μέχρι τώρα έχουμε περιγράψει τα perceptrons ως μια μέθοδο η οποία ζυγίζει τα στοιχεία για τη λήψη αποφάσεων. Ένας άλλος τρόπος όπου τα perceptrons μπορούν να χρησιμοποιηθούν είναι για να υπολογίσουν τις στοιχειώδεις λογικές λειτουργίες όπως οι AND, OR και NAND. Για παράδειγμα, ας υποθέσουμε ότι έχουμε ένα perceptron με δύο εισόδους, το καθένα με βάρος -2 -2, και μια τιμή κατωφλιού 3.



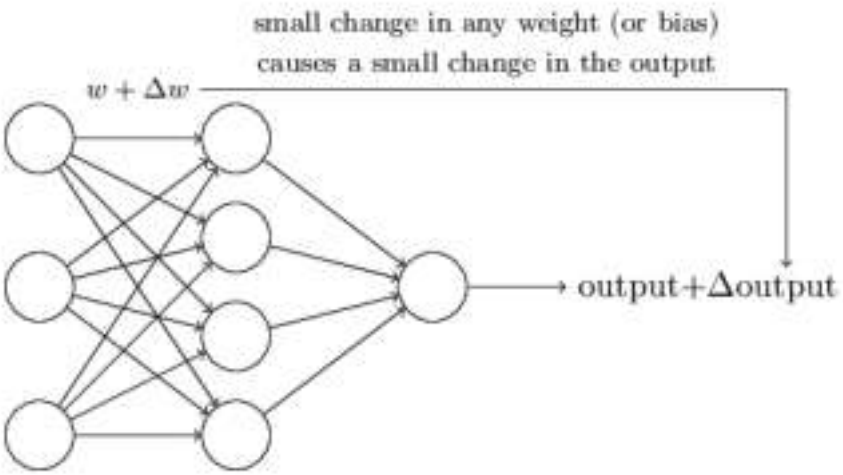
Σχήμα 8

Παρατηρούμε ότι αν θέσουμε ως τιμές εισόδου 0 0, τότε παράγεται ως έξοδο η τιμή 1, αφού (−2) ∗0 + (−2) ∗0 + 3 = 3 είναι θετικός αριθμός. Αντίστοιχα αν θέσουμε ως τιμές εισόδου 1 1, τότε παρατηρούμε ότι ως έξοδο παράγεται η τιμή 0, αφού (−2) ∗1 + (−2) ∗1 + 3 = −1 είναι αρνητικός αριθμός. Με αυτό τον τρόπο το συγκεκριμένο perceptron δημιουργεί μια NAND gate(2).

Το παραπάνω παράδειγμα μας δείχνει ότι μπορούμε να χρησιμοποιήσουμε τα perceptrons για να υπολογίσουμε τις απλές λογικές συναρτήσεις. Για την ακρίβεια μπορούμε να τα χρησιμοποιήσουμε για να υπολογίσουμε οποιαδήποτε λογική συνάρτηση. Ο λόγος είναι ότι μέσω μίας NAND gate μπορούμε να κατασκευάσουμε οποιονδήποτε υπολογισμό.

1.4.2Σιγμοειδής Συνάρτηση

Τα perceptrons ενώ είναι πολύ απλά και εύχρηστα, έχουν όμως ένα πρόβλημα όσον αφορά της τεχνικές εκμάθησης. Ας υποθέσουμε ότι έχουμε ένα δίκτυο perceptrons που θα θέλαμε να χρησιμοποιήσουμε και να το εκπαιδεύσουμε για να λύσουμε κάποιο πρόβλημα. Για να δούμε πώς μπορεί να λειτουργήσει η μάθηση, ας υποθέσουμε ότι κάνουμε μια μικρή αλλαγή σε κάποιο βάρος (ή στη πόλωση) στο δίκτυο. Αυτό που θα θέλαμε είναι αυτή η μικρή αλλαγή βάρους να προκαλέσει μόνο μια μικρή αντίστοιχη αλλαγή στην έξοδο από το δίκτυο. Αυτό που θέλουμε να δούμε, παρουσιάζεται σχηματικά στο σχήμα 9.



Σχήμα 9: By Michael Nielsen - Neural Networks and Deep Learning

Εάν αυτό ίσχυε όντως, ότι δηλαδή μία μικρή αλλαγή στα βάρη (ή στη πόλωση), θα προκαλούσε μόνο μία μικρή αντίστοιχα αλλαγή στην έξοδο του δικτύου, τότε θα μπορούσαμε να χρησιμοποιήσουμε το γεγονός αυτό για να τροποποιήσουμε τα βάρη (ή τις πολώσεις) ώστε να κάνουμε το δίκτυο να συμπεριφέρεται περισσότερο με τον τρόπο που θέλουμε. Αλλάζοντας τις τιμές στα βάρη (ή στις πολώσεις) ξανά και ξανά, θα μπορούσαμε να εκπαιδεύσουμε το δίκτυο με αυτόν τον απλό θεωρητικά τρόπο.

Το πρόβλημα όμως έγκειται στο γεγονός ότι όταν το δίκτυο περιέχει perceptrons, τότε μία μικρή αλλαγή στα βάρη (ή στις πολώσεις) οποιουδήποτε μεμονωμένου perceptron, μπορεί να προκαλέσει την έξοδο του δικτύου να αναστραφεί πλήρως. Αυτό καθιστά δύσκολο να δούμε πώς να τροποποιούμε σταδιακά τα βάρη και τις πολώσεις, έτσι ώστε το δίκτυο να πλησιάζει περισσότερο στην επιθυμητή συμπεριφορά.

Μπορούμε να ξεπεράσουμε αυτό το πρόβλημα με την εισαγωγή ενός νέου τύπου τεχνητού νευρώνα που ονομάζεται Σιγμοειδής Νευρώνας. Οι Σιγμοειδείς Νευρώνες είναι παρόμοιοι με τους perceptrons, αλλά τροποποιούνται έτσι ώστε οι μικρές αλλαγές στα βάρη τους και στη πόλωση, να προκαλούν μόνο 21 μικρή αλλαγή στην έξοδο του δικτύου. Αυτό είναι το κρίσιμο γεγονός που θα επιτρέψει σε ένα δίκτυο Σιγμοειδών Νευρώνων να εκπαιδευτούν.

Θα απεικονίσουμε σιγμοειδείς νευρώνες με τον ίδιο τρόπο που απεικονίσαμε perceptrons, στο σχήμα 9.

Ακριβώς όπως και με τα perceptrons, ο Σιγμοειδής Νευρώνας, έχει εισόδους x1, □2, … , □□, αλλά πλέον τώρα μπορούμε να δώσουμε τιμές εισόδου, οποιαδήποτε τιμή μεταξύ 0 και 1. Επίσης όπως και τα perceptrons ο Σιγμοειδής Νευρώνας, έχει βάρη w1, □2, … , □□για κάθε είσοδο και μία συνολική πόλωση, b. Οι τιμές εξόδου όμως δεν είναι πλέον αποκλειστικά 0 και 1. Αντίθετα είναι □(□∗□+ □), όπου το σ ονομάζεται σιγμοειδής συνάρτηση(3) και καθορίζεται από

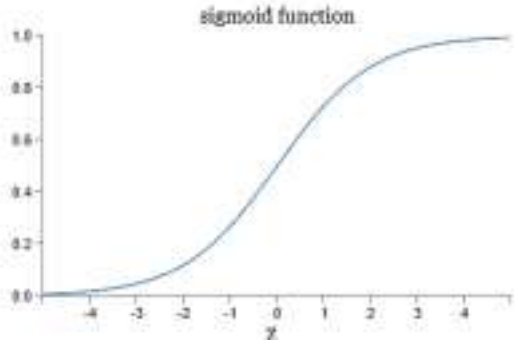
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

Για να το θέσουμε πιο γενικά, η τιμή εξόδου ενός σιγμοειδούς νευρώνα με εισόδους x1, □2, … , □□, βάρη w1, □2, … , □□και πόλωση □είναι

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

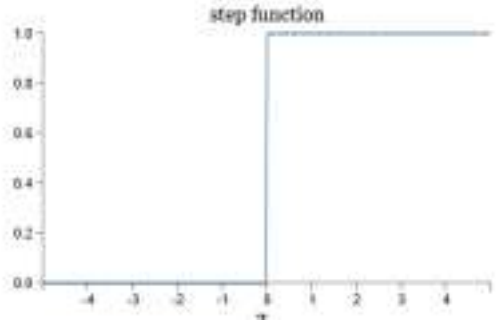
Εκ πρώτης όψεως, οι σιγμοειδείς νευρώνες εμφανίζονται πολύ διαφορετικοί από τους perceptrons. Στην πραγματικότητα, υπάρχουν πολλές ομοιότητες μεταξύ τους. Για να κατανοήσουμε την ομοιότητα αυτή, υποθέτουμε ότι z ≡ □∗□+ □είναι ένας μεγάλος θετικός αριθμός. Τότε □−□≈ 0 και επίσης □(□) ≈ 1. Με άλλα λόγια, η τιμή εξόδου ενός σιγμοειδή νεύρωνα σε αυτή την περίπτωση είναι προσεγγιστικά 1, όπως θα ήταν και στην περίπτωση ενός perceptron. Αντίστοιχα ισχύει και το ανάποδο, αν δηλαδή z ≡ □∗□+ □είναι ένας μεγάλος αρνητικός αριθμός, τότε η τιμή εξόδου ενός σιγμοειδή νευρώνα σε αυτή την περίπτωση είναι προσεγγιστικά 0, όπως θα ήταν και στην περίπτωση ενός perceptron.

Όμως, η αλγεβρική μορφή του σ, στην πραγματικότητα δεν είναι τόσο σημαντική, αυτό που είναι πραγματικά σημαντικό είναι η μορφή της συνάρτησης, όπως φαίνεται στο σχήμα 18,



Σχήμα 10: Σιγμοείδης Συνάρτηση

η οποία είναι μία εξομαλυμένη μορφή της βηματικής συνάρτησης(step function)



Σχήμα 11: Βηματική Συνάρτηση

Στην πραγματικότητα εάν ήταν μία βηματική συνάρτηση, τότε ο σιγμοειδής νεύρωνας θα ήταν ένα perceptron, αφού η τιμή εξόδου θα ήταν 0 ή 1, ανάλογα με το αν το □∗□+ □ήταν θετικό ή αρνητικό αντίστοιχα. Αυτό είναι ένα κρίσιμο συμπέρασμα στο οποίο καταλήγουμε ότι χρησιμοποιώντας την πραγματική □συνάρτηση παίρνουμε ένα εξομαλυμένο perceptron. Αυτό σημαίνει ότι μικρές αλλαγές στα βάρη και στη πόλωση θα μας δώσουν μικρές αλλαγές και στην τιμή εξόδου του δικτύου.

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j}\Delta w_j + \frac{\partial output}{\partial b}\Delta b$$

Η παραπάνω έκφραση λέει κάτι απλό και σημαντικό, ότι το □□□□□□□είναι μία γραμμική συνάρτηση των αλλαγών □□□και □□στα βάρη και στη πόλωση αντίστοιχα. Αυτή η γραμμικότητα, είναι που επιτρέπει μικρές αλλαγές στα βάρη και στη πόλωση να επιφέρουν μικρές αλλαγές στη τιμή εξόδου. Έτσι καταλήγουμε στο συμπέρασμα ότι ενώ οι σιγμοειδής νευρώνες έχουν την ίδια ποιοτική συμπεριφορά με τα

perceptrons, παρόλα αυτά καθιστούν πιο εύκολο να κατανοήσουμε πως μικρές αλλαγές στα βάρη ή στη πόλωση επιφέρουν και μικρές αλλαγές στην τιμή εξόδου του δικτύου.

Από τα παραπάνω συμπεραίνουμε ότι προφανώς μία μεγάλη διαφορά μεταξύ των perceptrons και των σιγμοειδών νευρώνων είναι ότι οι τελευταίοι μπορούν και παράγουν τιμές εξόδου οποιονδήποτε πραγματικό αριθμό μεταξύ 0 και 1. Αυτό όμως καμιά φορά μπορεί να μην είναι βολικό. Για παράδειγμα αν θέλουμε να αποφασίσουμε για ένα γεγονός αν ισχύει ή όχι, θα ήταν πιο βολικό το δίκτυο να έδινε τιμές 0 και 1. Αυτό στην πράξη μπορούμε να το προσπεράσουμε δημιουργώντας μία σύμβαση, όπου για παράδειγμα τιμές εξόδου μεγαλύτερες του 0.5 σημαίνει ότι αυτό το γεγονός ισχύει και αντίστοιχα μικρότερες ότι δεν ισχύει.

1.4.3 Η Υπερβολική Εφαπτομένη (Hyperbolic Tangent) Συνάρτηση Ενεργοποίησης

Αν και η Σιγμοειδής Συνάρτηση ενεργοποίησης έχει μία ωραία βιολογική ερμηνεία, αυτή μπορεί να προκαλέσει ένα νευρωνικό δίκτυο να κολλήσει κατά τη διάρκεια της εκπαίδευσής του. Αυτό οφείλεται εν μέρει στο γεγονός ότι εάν παρέχεται έντονα αρνητική τιμή εισόδου, εκπέμπει τιμές πολύ κοντά στο μηδέν και αυτό μπορεί να οδηγήσει στο ότι η ενημέρωση των παραμέτρων κατά τη διάρκεια της εκπαίδευσης, να μην γίνεται όσο τακτικά θέλουμε. (Stansbury, 2014)

Έτσι μία ενναλακτική συνάρτηση είναι η Υπερβολική Εφαπτομένη

$$\varphi_{tanh}(v) = \frac{\sinh(v)}{\cosh(v)} = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

Η συνάρτηση αυτή είναι επίσης μία Σιγμοειδής συνάρτηση όσον αφορά το σχήμα της, η διαφορά της όμως βρίσκεται στο γεγονός ότι παίρνει τιμές στο διάστημα (−1,1). Το γεγονός ότι οι τιμές εξόδου είναι κεντραρισμένες στο μηδέν, την κάνει προτιμότερη της Σιγμοειδής Συνάρτησης.

1.4.4 Ανορθωμένη Γραμμική Συνάρτηση Ράμπας (Rectified Linear Unit – ReLU)

Η συνάρτηση ράμπας έχει την εξής μορφή

$$\varphi(v) = \max(0, v)$$

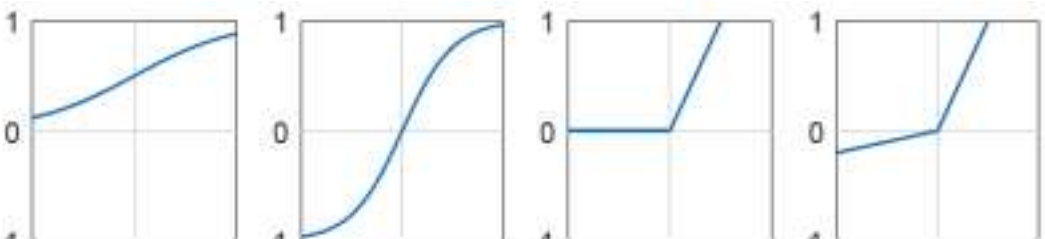
Η συγκεκριμένη συνάρτηση είναι η πιο δημοφιλής συνάρτηση ενεργοποίησης για Βαθιά Νευρωνικά Δίκτυα (DNN). Προτιμάται από τις υπόλοιπες συναρτήσεις διότι έχει την δυνατότητα να εκπαιδεύσει ένα δίκτυο αρκετά πιο γρήγορα, δίνοντας ακριβή αποτελέσματα. Ωστόσο, ένα σημαντικό μειονέκτημα της είναι πως κάποιες φορές μπορεί να οδηγήσει ορισμένους νευρώνες του δικτύου σε κάποιες τιμές βαρών, οι οποίες τους αποτρέπουν να ενεργοποιηθούν. Έτσι αυτοί οι νευρώνες νεκρώνουν, σταματάνε δηλαδή να εκπαιδεύονται.

Τη λύση στο πρόβλημα αυτό, δίνει η Παραμετροποιημένη Συνάρτηση Ράμπας (PReLU)

$$\varphi(v) = \begin{cases} v, & \text{αν } v > 0 \\ \alpha * v, & \text{διαφορετικά} \end{cases}$$

Η συνάρτηση αυτή πολλαπλασιάζει την τιμή εξόδου με μία μικρή τιμή □, στην περίπτωση που η τιμή εισόδου είναι αρνητική.

Αν □= 0, τότε η συνάρτηση μετατρέπεται σε ReLU, ενώ αν το □, πάρει μία μικρή και σταθερή τιμή ονομάζεται Leaky ReLU.



Σχήμα 12: Οι τέσσερις βασικές συναρτήσεις ενεργοποίησης.

1.4.5 Συνάρτηση Softmax

Όταν θέλουμε να αντιμετωπίσουμε προβλήματα κατηγοριοποίησης, οι προηγούμενες συναρτήσεις, δεν μπορούν να μας βοηθήσουν πολύ. Για παράδειγμα, η σιγμοειδής συνάρτηση, μπορεί να χειριστεί μέχρι δύο κλάσεις, κάτι το οποίο πολύ συχνά δεν μας είναι αρκετό. Η συνάρτηση Softmax, η οποία είναι μία γενίκευση της λογιστικής συνάρτησης, μπορεί να μας βοηθήσει σε αυτό το πρόβλημα. Χρησιμοποιείται συχνά στο τελευταίο στρώμα ενός δικτύου, στο οποίο όπως γνωρίζουμε προκύπτουν οι τιμές εξόδουτου δικτύου και λειτουργεί συμπυκνώνοντας τις τιμές αυτές, έτσι ώστε να είναι μεταξύ 0 και 1, αλλά και το άθροισμα τους να ισούται με τη μονάδα. Έτσι κάθε τιμή εξόδου, που προκύπτει από μία συνάρτηση softmax, είναι ισοδύναμη με μία κατηγορική συνάρτηση πιθανότητας. Η συνάρτηση αυτή, χρησιμοποιείται ευρέως στα βαθιά νευρωνικά δίκτυα, στα οποία θα αναφερθούμε στη συνέχεια.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

1.5 Συνελικτικά Νευρωνικα Δικτύα (Convolution Neural Networks)

1.5.1 Αβαθή και Βαθιά Νευρωνικά Δίκτυα

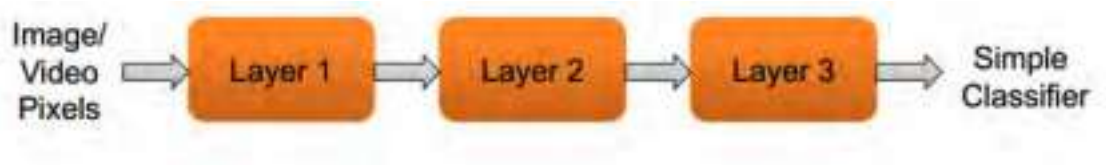
Τα Συνελικτικά Νευρωνικά Δίκτυα (ΣΝΔ) (Convolutional Neural Networks CNN), χωρίζονται σε δυο μεγάλες κατηγορίες τα Αβαθή Νευρωνικά Δίκτυα (Shallow Neural Networks) και τα Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks). Ένα συνελικτικό επίπεδο (convolutional layer) είναι ουσιαστικά ένα σύνολο από νευρώνες που εκτελούν συνέλιξη των φίλτρων που έχουν προκαθοριστεί, με την εικόνα-διάνυσμα που δέχονται στην είσοδο. Κάθε επίπεδο μπορεί να περιλαμβάνει νευρώνες που εκτελούν συνέλιξη, διαδικασίες pooling, εισαγωγή μη γραμμικότητας ή ακόμη και κανονικοποίηση, ενώ έχει διακριτές εισόδους και εξόδους. Οι διαστάσεις των φίλτρων που περιλαμβάνουν, ο αριθμός τους και το βάθος τους (αριθμός καναλιών) μπορεί να διαφέρει σημαντικά ανάλογα με το πρόβλημα.

1.5.2Καθιερωμένη Προσέγγιση Αναγνώρισης

Κατά τον καθιερωμένο τρόπο αναγνώρισης αντικειμένων σε εικόνα ή βίντεο, εικόνες ή αλληλουχία εικόνων (βίντεο) δίνονται σαν είσοδο σε κάποιο σύστημα προκειμένου να εκτελεστεί ο αλγόριθμος αναγνώρισης. Συνήθως δε, τα χαρακτηριστικά που χρησιμοποιούνται για την αναγνώριση είναι κατά βάση χαρακτηριστικά επιλεγμένα από ανθρώπινο παράγοντα, δηλαδή δεν μεσολαβεί κάποια διαδικασία εκμάθησης μέσω της οποίας εκπαιδεύεται ένας ταξινομητής για την εκπλήρωση του σκοπού της ταξινόμησης. Έτσι, από νωρίς έγινε γνωστό ότι το κλειδί στην αναγνώριση και ταξινόμηση, είναι η επιλογή σωστών χαρακτηριστικών και η καλή περιγραφή τους με τις πιο κλασικές πλέον μεθόδους όπως π.χ. SIFT, HOG κλπ. Το ερώτημα που προέκυψε λοιπόν, συνοψίζεται στην ανάγκη για εύρεση μιας διαδικασίας εκμάθησης αυτών των χαρακτηριστικών, χωρίς την ανάγκη παρέμβασης του ανθρώπινου παράγοντα. Μια καλή προσέγγιση και λύση αυτού του προβλήματος, δίνουν τα CNN μέσω της εξαγωγής χαρακτηριστικών από ενδιάμεσα κρυφά επίπεδα.

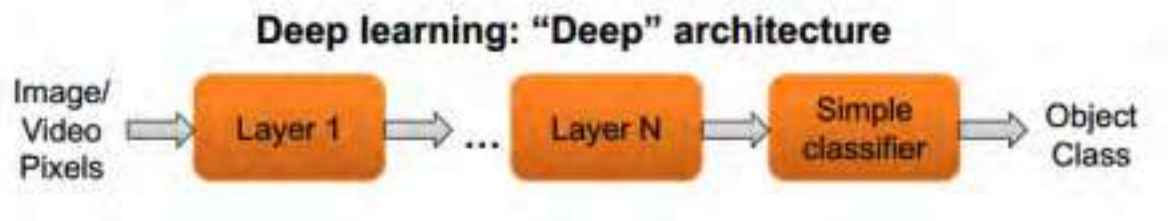
Κάθε Νευρωνικό Δίκτυο,αποτελείται από διάφορα επίπεδα τα οποία περιλαμβάνουν τις αντίστοιχες εισόδους και εξόδους. Εφόσον μετά από κάθε τέτοιο επίπεδο η έξοδος αποτελεί μια αναπαράσταση της εισόδου με διαφορετικές (συνήθως μικρότερες) διαστάσεις, μπορούμε να θεωρήσουμε κάθε έξοδο ενός συνελικτικού νευρώνα σαν χαρακτηριστικό διάνυσμα που περιγράφει την αντίστοιχη είσοδο. Το ίδιο μπορεί να υποτεθεί για την έξοδο ενός pooling νευρώνα, η οποία αποτελεί την προέκταση μιας διαδρομής από την είσοδο μιας εικόνας προς το κρυφό επίπεδο συνέλιξης το οποίο

εξετάζουμε. Από τη θεωρία γνωρίζουμε ότι τα βάρη στις εισόδους κάθε κρυμμένου επιπέδου πρέπει οπωσδήποτε να είναι τυχαία και επομένως για κάθε είσοδο x, η έξοδος ενός συγκεκριμένου νευρώνα θα είναι μοναδική για κάθε εικόνα. Τα χαρακτηριστικά διανύσματα δηλαδή που προκύπτουν ακόμη και από το ίδιο κρυφό επίπεδο, διαφέρουν μεταξύ τους.



Σχήμα 13: Deep Neural Network, 3 Layers.

Στα Deep Neural Networks, ο αριθμός των συνελίξεων μπορεί να είναι σημαντικά αυξημένος και τα χαρακτηριστικά διανύσματα που προκύπτουν να έχουν ελαττωμένες τις διαστάσεις σε μεγάλο ποσοστό, χωρίς όμως να χάνουν την ουσία της πληροφορίας που περιλαμβάνουν. Γενικά, όσο πιο “βαθύ” είναι το επίπεδο το οποίο εξετάζουμε, τόσο πιο “γενικά” (global) είναι τα χαρακτηριστικά και πιο αναλλοίωτα. Παρατηρούμε ότι στο τέλος του Δικτύου περιλαμβάνεται πάντα ένας ταξινομητής. Στην εικόνα 26, φαίνεται η γενική εικόνα μιας Deep Architecture, συγκρινόμενη με την προηγούμενη εικόνα ως προς τον αριθμό των επιπέδων.



Σχήμα 14: Deep Neural Network, N layers.

1.6 Συστατικά Μέρη

Υπάρχουν 4 βασικές λειτουργίες που εκτελούνται στα ΣΝΔ:

1. Συνέλιξη (Convolution)
2. Μη γραμμικότητα (Non Linearity) - ReLU
3. Συγκέντρωση ή Υπο-Δειγματοληψία (Pooling / Sub sampling)

4. Κατηγοριοποίηση από πλήρως συνδεδεμένο επίπεδο (Fully Connected Layer for Classification)

1.6.1 Πλήρως συνδεδεμένα δίκτυα

Κατά την εκπαίδευση ενός CNN, πολλές φορές ερχόμαστε αντιμέτωποι με το πρόβλημα της πολυπλοκότητας. Για παράδειγμα, εάν θέλουμε να εκπαιδεύσουμε ένα CNN με εικόνες 8x8 ή μέχρι και 28x28 pixels, μια αρχιτεκτονική “πλήρως συνδεδεμένη” ή αλλιώς “fully connected” όπου κάθε νευρώνας των κρυφών επιπέδων συνδέεται με την έξοδο όλων των νευρώνων του προηγούμενου επιπέδου, είναι αρκετά απλή και λογική, ενώ είναι υπολογιστικά εφικτό να μάθουμε χαρακτηριστικά για όλη την εικόνα. Ωστόσο, σε μεγαλύτερες εικόνες, όπως για παράδειγμα 96x96 pixels, η εκμάθηση χαρακτηριστικών για όλη την εικόνα με “fully connected” αρχιτεκτονική είναι πολύ δύσκολη, αφού έχουμε περίπου 10^4 εισόδους, ενώ αν σκεφτούμε ότι θέλουμε να μάθουμε 100 χαρακτηριστικά προκύπτουν αμέσως 10^6 παράμετροι για εκμάθηση. Οι feedforward και backpropagation αλγόριθμοι θα είναι περίπου 10^2 φορές πιο αργοί σε σχέση με τις 28x28 εικόνες.

1.6.2Τοπικά συνδεδεμένα δίκτυα

Μια απλή λύση, είναι αυτή των τοπικά συνδεδεμένων δικτύων (ή αλλιώς Locally Connected Networks), κατά τα οποία μπορούμε να περιορίσουμε τις συνδέσεις μεταξύ των νευρώνων εισόδου και των κρυφών νευρώνων, επιτρέποντας κάθε κρυφό νευρώνα να συνδέεται μόνο με ένα μικρό μέρος των νευρώνων εισόδου.

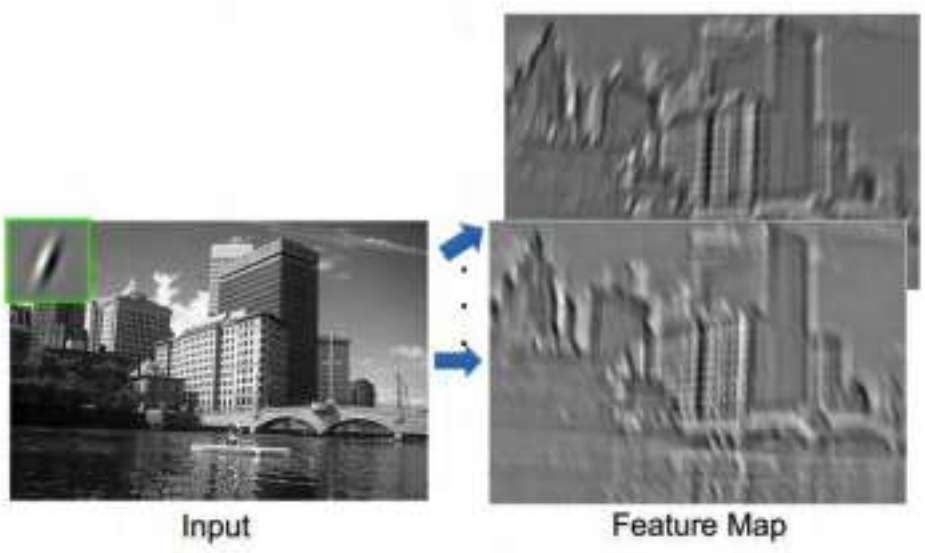
Πιο συγκεκριμένα, κάθε κρυφός νευρώνας θα συνδέεται μόνο με ένα μικρό μέρος συνεχών pixels εισόδου. Το ίδιο συμβαίνει και σε άλλες περιπτώσεις σημάτων όπως ο ήχος· ένας νευρώνας εισόδου θα συνδέεται με ένα συγκεκριμένου χρόνου και διάρκειας δείγματος ήχο.

Η συγκεκριμένη αρχιτεκτονική είναι εμπνευσμένη από τον τρόπο σύνδεσης του πρώιμου οπτικού συστήματος που απαντάται στη βιολογία, όπου οι νευρώνες στον οπτικό φλοιό έχουν εντοπισμένους δεκτικούς τομείς (πχ αποκρίνονται μόνο σε ερεθίσματα σε μια συγκεκριμένη τοποθεσία).

1.6.3Συνελίξεις

Η κεντρική ιδέα στην οποία βασίζονται τα CNN, είναι ότι τα χαρακτηριστικά ενός patch εικόνας, είτε αυτά αφορούν κλίσεις, είτε ακμές κλπ, είναι τοπικά εντοπισμένα. Μερικά από τα πλεονεκτήματα των CNN, είναι η ανεξαρτησία τους ως προς τις μετατοπίσεις της εικόνας σε διάφορες διευθύνσεις (Translation Invariance), ο μικρός αριθμός παραμέτρων (βάρη φίλτρων) ή ακόμη και το μεγάλο βήμα μεταξύ των patches που χρησιμοποιούνται κατά το pooling, το οποίο έχει ως αποτέλεσμα γρηγορότερη εκτέλεση αλγορίθμου και μικρότερες απαιτήσεις μνήμης.

Στην εικόνα 15, φαίνεται στα αριστερά η αρχική εικόνα μαζί με το φίλτρο που θα εφαρμόσουμε επ' αυτής και στα δεξιά το αποτέλεσμα μετά την εφαρμογή του με χρήση συνέλιξης.

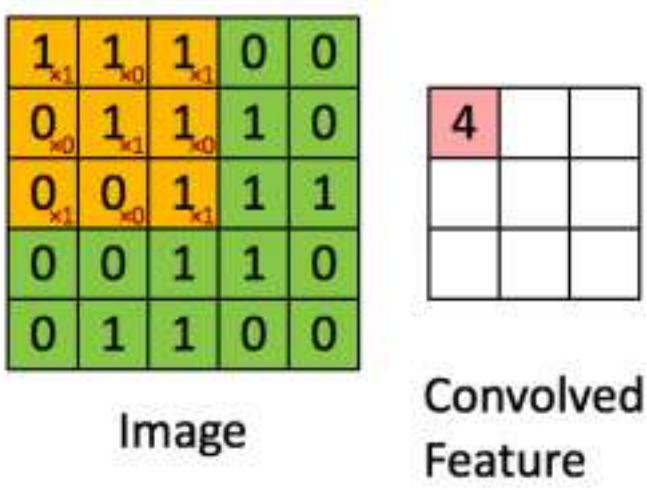


Σχήμα 15: Αριστερά η αρχική εικόνα και το φίλτρο, δεξιά αποτέλεσμα συνέλιξης του φίλτρου με την εικόνα.

Από τη θεωρία γνωρίζουμε ότι οι φυσικές εικόνες έχουν την ιδιότητα της "στατικότητας", δηλαδή η στατιστική που διέπει ένα patch (παράθυρο) της εικόνας είναι η ίδια σε κάθε άλλο μέρος της. Επομένως τα χαρακτηριστικά που μαθαίνουμε για κάποιο patch της εικόνας μπορούν να εφαρμοστούν σε κάθε άλλο μέρος της και μπορούμε να χρησιμοποιήσουμε τα ίδια χαρακτηριστικά για όλα τα patch.

Ακριβέστερα, έχοντας μάθει χαρακτηριστικά για μικρά patches (ας πούμε 8x8) τυχαία επιλεγμένα από μια ευρύτερη εικόνα, μπορούμε να εφαρμόσουμε αυτό τον 8x8 ανιχνευτή χαρακτηριστικών παντού. Αυτό γίνεται παίρνοντας τα 8x8 χαρακτηριστικά και εκτελώντας συνέλιξη πάνω στην εικόνα με αυτά, παρατηρώντας τις αποκρίσεις ή αλλιώς τα activations σε διαφορετικές τοποθεσίες της εικόνας.

Για να δώσουμε ένα παράδειγμα, ας υποθέσουμε ότι έχουμε εκμάθει χαρακτηριστικά για 8x8 patches από μια 96x96 εικόνα. Ας υποθέσουμε ακόμη ότι αυτό έχει γίνει με έναν Autoencoder (θα εξηγήσουμε πιο κάτω την έννοια αυτή) και ότι αυτός έχει 100 κρυφούς νευρώνες. Για να πάρουμε τα χαρακτηριστικά για κάθε 8x8 patch από την 96x96 εικόνα αρχίζοντας από τα (1,1), (1,2), … , (89,89), θα περάσουμε το patch πάνω στην εικόνα μέσω ενός sparse Autoencoder και θα λάβουμε 100 sets των 89x89 χαρακτηριστικών.



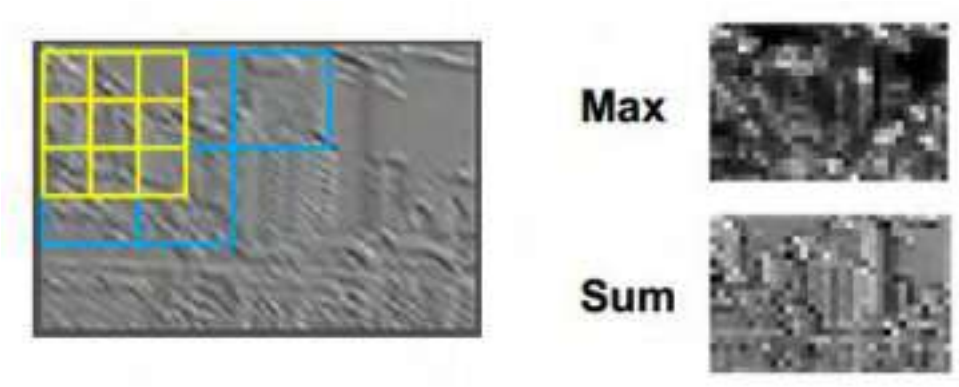
Σχήμα 16: Διαδικασία συνέλιξης.

Δεδομένων μερικών μεγάλων διαστάσεων (r x c) εικόνων x_{large}, αρχικά εκπαιδεύουμε έναν **sparse** Autoencoder σε μικρά a x b patches x_{small} μεγάλων εικόνων, μαθαίνοντας k χαρακτηριστικά $f = \sigma(W^{(1)}x_{small} + b^{(1)})$όπου σ είναι η sigmoid και $W^{(1)}$τα βάρη, $b^{(1)}$τα bias από τους νευρώνες εισόδου στους κρυφούς νευρώνες. Για κάθε a x b patch x_s της μεγάλης εικόνας, υπολογίζουμε την $f_s = \sigma(W^{(1)} x_s + b^{(1)})$ δίνοντας τα $f_{convolved}$, έναν k x (r - a + 1) x (c - b + 1) πίνακα **convolved** χαρακτηριστικών.

Ακολουθεί περιγραφή για το πώς κάνουμε “pool” για να λάβουμε καλύτερα αποτελέσματα.[1][3][7][34] reference nemertes

1.6.4 Συγκέντρωση (Pooling)

Τα επίπεδα συγκέντρωσης (pooling layers) στα CNNs, συνοψίζουν τις εξόδους γειτονικών γκρουπ νευρώνων εντός ενός παραθύρου (patch) με μια αντιπροσωπευτική τιμή, ενώ συνήθως τα γειτονικά παράθυρα δεν επικαλύπτονται. Πρόκειται ουσιαστικά για μια διαδικασία υπο-δειγματοληψίας των δεδομένων ενώ για καλύτερη κατανόηση της διαδικασίας μπορούμε να φανταστούμε ένα επίπεδο pooling σαν ένα “πλέγμα” pooling νευρώνων τοποθετημένων σε απόσταση *S*pixels, καθένας από τους οποίους συνοψίζει μια περιοχή *ZxZ* με κέντρο τον ίδιο τον νευρώνα. Θέτοντας*S=Z* λαμβάνουμε τα κλασικά αποτελέσματα του κοινώς χρησιμοποιούμενου, μη επικαλυπτόμενων παραθύρων pooling, ενώ για *S<Z*λαμβάνουμε τιμές από επικαλυπτόμενα παράθυρα. Το pooling αποτελεί μια πολύβασική λειτουργία για κάθε CNN, αφού απλοποιεί πολύ τη διαδικασία λόγω της σημαντικής μείωσης των δεδομένων κι επομένως του αριθμού των απαιτούμενων πράξεων. Οι επικρατέστερες κατηγορίες του pooling είναι το max, sum και average pooling, ενώ μπορεί τα παράθυρα που χρησιμοποιούνται να επικαλύπτονται ή και όχι ανάλογα με τις ανάγκες του προβλήματος. Η διαδικασία του pooling, εκτός απ τη μείωση του μεγέθους των δεδομένων, μας δίνει τη δυνατότητα προσθήκης περισσότερης πληροφορίας στην αρχική εικόνα μέσω των αρχικών διαστάσεων ενώ είναι ανεξάρτητο μικρών μετασχηματισμών.



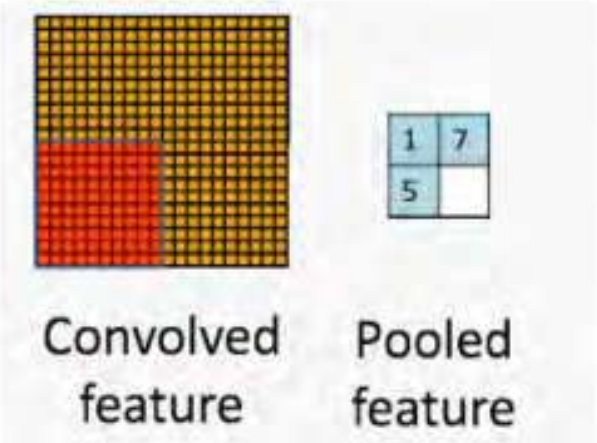
Σχήμα 17: 9 Αποτέλεσμα εφαρμογής max pooling και sum pooling πάνω σε εικόνα.

Όταν έχουμε πλέον λάβει τα χαρακτηριστικά διανύσματα που περιγράψαμε νωρίτερα, αποφασίζουμε για το μέγεθος των παραθύρων, ας πούμε *MxN*που θα κάνουμε pool τα δεδομένα. Τότε διαιρούμε τα χαρακτηριστικά μας σε *MxN* περιοχές και παίρνουμε το μέγιστο (max) ή τον μέσο όρο (mean) του παραθύρου, το οποίο αποτελεί το νέο χαρακτηριστικό μας. Αυτές οι pooled περιοχές μπορούν πλέον να χρησιμοποιηθούν για ταξινόμηση.

Μετά την διαδικασία του pooling και αφού έχουμε λάβει τα χαρακτηριστικά, συνήθως θέλουμε να τα χρησιμοποιήσουμε για ταξινόμηση. Στη θεωρία, μπορούμε να χρησιμοποιήσουμε όλα τα χαρακτηριστικά σε έναν ταξινομητή όπως ο softmax classifier, ωστόσο αυτό είναι υπολογιστικά αδύνατο. Ας αναλογιστούμε την περίπτωση εικόνων 96x96 pixels και 400 χαρακτηριστικά από 8x8 inputs. Κάθε συνέλιξη φέρει ως αποτέλεσμα μια έξοδο μεγέθους (96-8+1) x (96-8+1) = 7921, και εφόσον έχουμε 400 χαρακτηριστικά, προκύπτει ένα διάνυσμα 89^{2}* 400 = 3,168,400 χαρακτηριστικά ανά δείγμα. Η εκμάθηση ενός ταξινομητή με 3+ εκατομμύρια χαρακτηριστικά, μπορεί να γίνει πολύ επίπονη και υπέρ-εξειδικευμένη (over-fitting).

Προκειμένου να το διευθετήσουμε, ας ανακαλέσουμε το γεγονός ότι αποφασίσαμε να λαμβάνουμε convolved εικόνες επειδή έχουν την ιδιότητα της “στατικότητας”, δηλαδή ότι τα χαρακτηριστικά που είναι χρήσιμα σε μια περιοχή της εικόνας, είναι χρήσιμα σε κάθε άλλη περιοχή της. Έτσι, προκειμένου να περιγράψουμε μια μεγάλη εικόνα θα λάβουμε ένα μίγμα στατιστικών των [65] χαρακτηριστικών των παραπάνω εικόνων. Για παράδειγμα, θα μπορούσαμε να λάβουμε έναν μέσο όρο ή τη μέγιστη τιμή ενός συγκεκριμένου χαρακτηριστικού για ένα patch της εικόνας. Έτσι, ελαττώνονται σημαντικά οι διαστάσεις των δεδομένων και πετυχαίνουμε μικρότερο over-fitting. Ονομάζουμε αυτή τη μέθοδο “pooling”, ή πιο συγκεκριμένα “mean pooling” και “max pooling” ανάλογα τη μέθοδο που χρησιμοποιούμε.

Στην εικόνα 18 φαίνεται η διαδικασία του pooling:

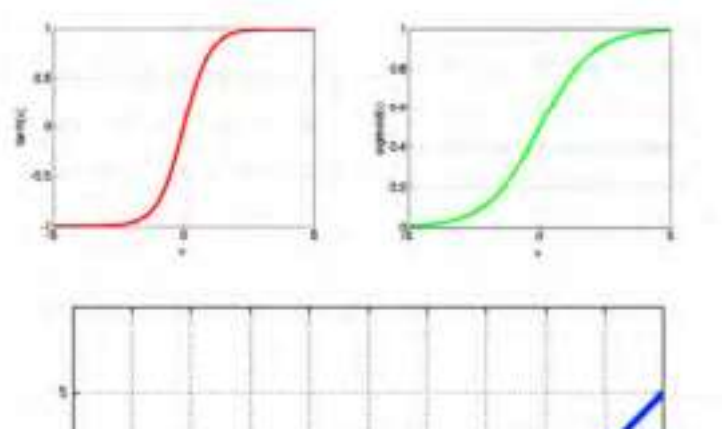


Σχήμα 18: 9 Pooling διαδικασία.

Εάν κάποιος επιλέξει οι περιοχές που γίνεται το pooling να είναι συνεχόμενες στην εικόνα και λαμβάνει χαρακτηριστικά μόνο από τους ίδιους κρυφούς νευρώνες, τότε αυτοί οι pooling νευρώνες θα γίνουν “translation invariant” ή αλλιώς ανεξάρτητα μετατοπίσεων της εικόνας. Αυτό σημαίνει ότι το ίδιο χαρακτηριστικό θα ενεργοποιείται ακόμα και όταν η εικόνα υπόκειται μικρές μετατοπίσεις. Συνήθως γι αυτό το λόγο επιδιώκεται τα χαρακτηριστικά μας να είναι ανεξάρτητα μετατοπίσεων, όπως για παράδειγμα στην ανίχνευση αντικειμένων ή την αναγνώριση ήχου.

1.6.5 Μη γραμμικότητα

Η εισαγωγή της μη γραμμικότητας (non-linearity) δίνει σοβαρό πλεονέκτημα στα CNN έναντι άλλων γνωστών μεθόδων αντιμετώπισης πολλών προβλημάτων. Σαν παράδειγμα αναφέρουμε την περίπτωση ενός μη γραμμικού συστήματος και την προσπάθεια πρόβλεψης αυτού, όπου οι γραμμικές μέθοδοι αδυνατούν να δώσουν καλά αποτελέσματα, ιδιαίτερα όταν το σύστημα εμφανίζει χαοτική συμπεριφορά. Έτσι, γίνεται αμέσως αντιληπτό το πλεονέκτημα των μη γραμμικών ΝΔ έναντι άλλων γνωστών γραμμικών μεθόδων. Μερικά παραδείγματα μη γραμμικότητας αποτελούν η*tanh*, η *sigmoid* και η ευρέως χρησιμοποιούμενη Rectified Linear Unit (ReLU), οι μορφές των οποίων φαίνονται στο πιο κάτω σχήμα (εικόνα 19)



Σχήμα 19: Συναρτήσεις εισαγωγής μη γραμμικότητας. Με κόκκινη συμβολίζεται η tanh, με πράσσινο η sigmoid και με μπλε η ReLU.

Πιο κατάλληλη θεωρείται η ReLU, για το λόγο ότι απλοποιεί τον αλγόριθμο backpropagation και τον επιταχύνει, εφόσον οι πράξεις που χρειάζεται να εκτελεστούν είναι πολύ λιγότερες.[3] nemertes reference

1.6.6 Dropout

Υπάρχουν διάφορες μέθοδοι για να μειώσουμε τα σφάλματα εκπαίδευσης ενός Νευρωνικού Δικτύου, όπως για παράδειγμα να συγκρίνουμε τις προβλέψεις πολλών και διαφορετικών μοντέλων. Για μεγάλα Νευρωνικά Δίκτυα των οποίων η εκπαίδευση μπορεί να κρατήσει αρκετές ημέρες αυτή η μέθοδος είναι χρονοβόρα και σε ορισμένες περιπτώσεις ίσως και αδύνατη. Προς αντιμετώπιση αυτού του προβλήματος έχει αναπτυχθεί κάποια μέθοδος σύγκρισης μοντέλων αρκετά αποτελεσματική, το υπολογιστικό κόστος της οποίας είναι πολύ χαμηλό. Η μέθοδος αυτή, η οποία ονομάζεται “Dropout” συνίσταται στην την ανάθεση ως “0” της εξόδου κάθε κρυφού νευρώνα με πιθανότητα 0.5.

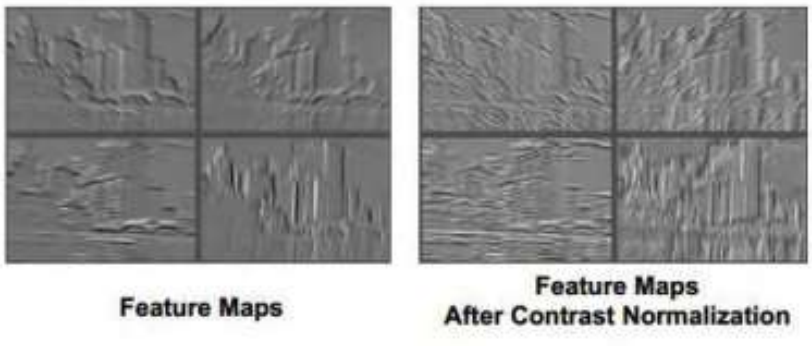
Οι νευρώνες που συμμετέχουν με αυτό τον τρόπο στο “Dropout” δεν συνεισφέρουν στη διάδοση προς τα εμπρός των σημάτων εκπαίδευσης και δεν συμμετέχουν στην διαδικασία του back-propagation. Έτσι, κάθε φορά που μια είσοδος

παρουσιάζεται στο Δίκτυο, εκείνο χρησιμοποιεί διαφορετική αρχιτεκτονική αλλά όλες αυτές οι αρχιτεκτονικές μοιράζονται τα ίδια βάρη. Αυτή η τεχνική μειώνει τις περίπλοκες συν-προσαρμογές των νευρώνων κι έτσι το Δίκτυο αποκτά τη δυνατότητα εκμάθησης ισχυρότερων χαρακτηριστικών.

Κατά τη διαδικασία της επαλήθευσης (test) χρησιμοποιούμε όλους τους νευρώνες πολλαπλασιάζοντας όμως τις εξόδους τους με 0.5 προκειμένου να [67] πάρουμε τον γεωμετρικό μέσο των κατανομών πρόβλεψης. Τέλος, αναφέρουμε ότι η χρήση του Dropout σχεδόν διπλασιάζει τον αριθμό των επαναλήψεων που απαιτούνται για σύγκλιση.[3] nemertes reference

1.6.7Κανονικοποίηση

Η κανονικοποίηση (Normalization), μπορεί να εφαρμοσθεί εντός ενός χαρακτηριστικού ή και μεταξύ ενός συνόλου χαρακτηριστικών και προσφέρει μικρότερη διακύμανση στα δεδομένα. Επίσης μπορεί να εφαρμοσθεί πριν ή μετά το pooling χωρίς ιδιαίτερη διαφορά μεταξύ των περιπτώσεων, ενώ δεν είναι απαραίτητη πάντοτε.[3] nemertes reference



Σχήμα 20: Εφαρμογή κανονικοποίησης σε convolved εικόνα.

1.6.8 Προ-εκπαίδευση

Κατά την εκπαίδευση ενός Συνελικτικού Νευρωνικού Δικτύου, η διαδικασία της οπισθοδιάδοσης σφάλματος συνίσταται στη διάδοση κλίσεων στα προηγούμενα επίπεδα του Δικτύου για τη διόρθωση των βαρών των φίλτρων που έχουν υπολογιστεί. Μερικά από τα πρώτα επίπεδα ενός πολύ-επίπεδου Νευρωνικού Δικτύου αλλάζουν ελάχιστα την τιμή

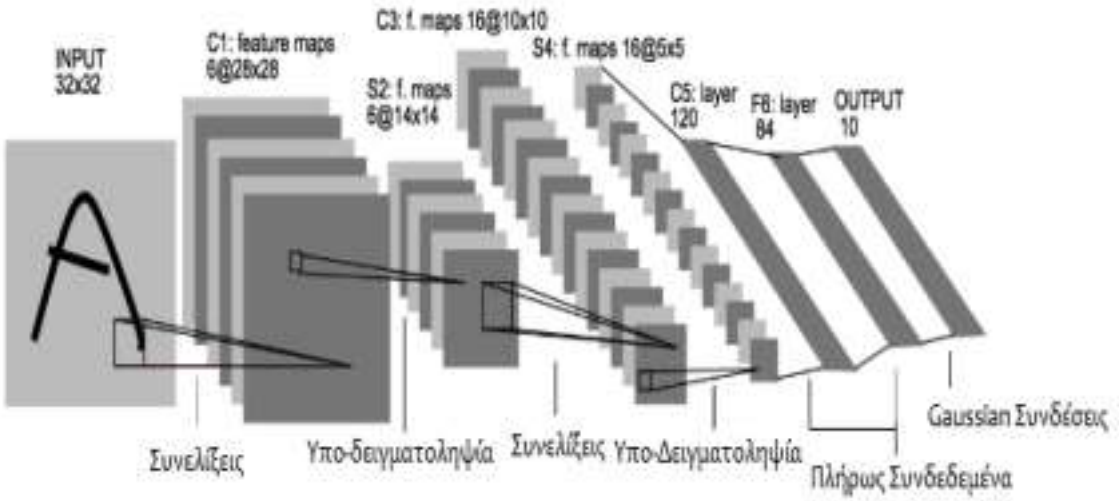
τους λόγω της μειωμένης διάδοσης της κλίσης (εκπαιδεύονται ανεπαρκώς) και αυτό μπορεί να αποτελέσει σημαντικό πρόβλημα. Η λύση έρχεται από την διάθεση ενός μεγάλου αριθμού δειγμάτων, τουλάχιστον 5000 δείγματα ανά κλάση και μετά από αρκετό χρόνο εκπαίδευσης το δίκτυο καταφέρνει να αποκτήσει την τελική και αποδοτική του μορφή.

Όταν τα δεδομένα εκπαίδευσης είναι ανεπαρκή, απαιτείται προ-εκπαίδευση και η τεχνική που ακολουθείται είναι η παροχή στο Δίκτυο διανυσμάτων-εικόνων μεγάλης βάσης δεδομένων από τομέα σχετικό με την εφαρμογή που θέλουμε να δημιουργήσουμε, ώστε να επέλθει σύγκλιση των παραμέτρων του Δικτύου. Στη συνέχεια, κατά την εκπαίδευση, δίνονται ως είσοδοι εικόνες-διανύσματα από τα δεδομένα που έχουμε στη διάθεσή μας. Στην περίπτωση των εικόνων, ένα επιπλέον βήμα που μπορεί να ακολουθηθεί, είναι η επέκταση της βάσης δεδομένων μέσα από επεξεργασία των εικόνων εκμεταλλευόμενοι του στοιχείου της “τοπικότητας” των Συνελικτικών Νευρωνικών Δικτύων. Έτσι, για κάθε εικόνα μπορούμε να [68] δημιουργήσουμε μερικά ελαφρώς παραλλαγμένα αντίγραφα, όπου έχουμε αλλάξει τις τιμές αριθμού τυχαίων pixel πάνω στην εικόνα, πχ ανάθεση σε ορισμένα pixel της τιμής “0” ή “1”, περιστροφές των εικόνων για ελάχιστες μοίρες (-10 εως +10 μοίρες) και μετατόπιση των εικόνων σε δυο διευθύνσεις με βήμα ένα μικρό αριθμό pixels (translation). Με αυτό τον τρόπο, δίνουμε στο Δίκτυο μια άλλη “οπτική” της ίδιας πληροφορίας, όπως συμβαίνει και στις εικόνες RGB όπου το Δίκτυο έχει στη διάθεσή του τρείς διαφορετικές εκδοχές της ίδιας εικόνας, με αποτέλεσμα η εκπαίδευσή του να γίνεται πιο αποτελεσματική.

$$\mathrm{Verif}(f_i, f_j, y_{ij}, \theta_{ve}) = \begin{cases} \frac{1}{2}\|f_i - f_j\|_2^2 & \text{if } y_{ij} = 1 \\ \frac{1}{2}\max\left(0, m - \|f_i - f_j\|_2\right)^2 & \text{if } y_{ij} = -1 \end{cases},$$

Η προ-εκπαίδευση γενικά κάνει την βελτιστοποίηση του Δικτύου εύκολη και μειώνει την υπερ-προσαρμογή (overfitting)

Σε άλλες μορφές νευρωνικών δικτύων, για παράδειγμα στους Autoencoders και τις RBMs, η προ-εκπαίδευση είναι αρκετά σημαντική αφού ένα από τα θεμελιώδη ζητήματα για την ορθή εκπαίδευσή τους αποτελεί η ανάθεση κατάλληλων αρχικών τιμών των βαρών τους.



1.7 Συνελικτικά Νευρωνικά Δίκτυα και η εξέλιξη τους στον χρόνο

1.7.1LeNet-5

Το μοντέλο LeNet-5 [1] pergamos reference αναπτύχθηκε το 1998 για να ταυτοποιεί χειρόγραφα ψηφία ταχυδρομικών κωδίκων σε υπηρεσίες ταχυδρομίου. Αυτό το παγιωμένο μοντέλο μας σύστησε τα ΣΝΔ με το τρόπο που τα ξέρουμε σήμερα.

Σχήμα 21: Σχηματική απεικόνιση του μοντέλου LeNet

Τα συνελικτικά επίπεδα χρησιμοποιούν ένα υποσύνολο από των καναλίων του προηγούμενου επιπέδου για κάθε φίλτρο, για να μειωθούν οι υπολογισμοί και να αναγκάσει ένα σπάσιμο της συμμετρίας στον δίκτυο. Τα επίπεδα υπο-δειγματοληψίας χρησιμοποιούν μια μορφή μέσης συγκέντρωσης (average pooling). Οι παράμετροι που χρησιμοποιεί αυτό το δίκτυο ανέρχονται στους 60.000.

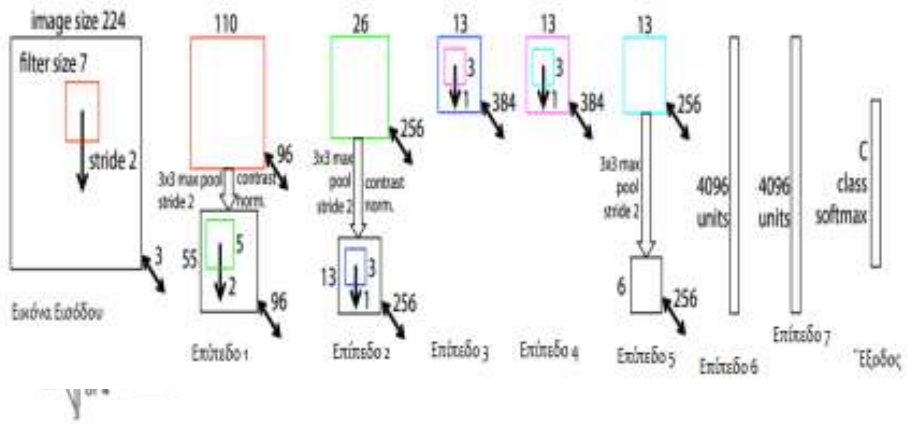
1.7.2AlexNet

Το AlexNet [2] αναπτύχθηκε από τον Alex Krizhevsky το 2012 για να λάβει μέρος στο διαγωνισμό ImageNet. Η γενική αρχιτεκτονική είναι αρκετά παρόμοια με αυτή του LeNet-5 μόνο που αυτό το μοντέλο είναι αισθητά μεγαλύτερο. Η επιτυχία αυτού του μοντέλου, που κέρδισε την πρώτη θέση στο διαγωνισμό ImageNet το 2012, έκανε πολλούς από αυτούς που ασχολούνται με τη μηχανική όραση να λάβουν σοβαρά υπόψιν τη βαθιά εκμάθηση ως μια πολύ καλή μέθοδο για ζητήματα σχετικά με το αντικείμενο τους. Το AlexNet διαθέτει 60 εκατομμύρια παραμέτρους και η δομή του φαίνεται στην παρακάτω εικόνα.

Σχήμα 22: Σχηματική απεικόνιση του μοντέλου AlexaNet

1.7.3ZFNet

Το μοντέλο αυτό [5] δημιουργήθηκε από τους Matthew Zeiler και Rob Fergus με το οποίο νίκησαν στο διαγωνισμό LSVRC του ImageNet το 2013. Αυτό το μοντέλο αποτελεί ουσιαστικά μια βελτίωση του AlexNet με κάποιες μικροαλλαγές στις παραμέτρους της αρχιτεκτονικής. Γενικότερα επέκτειναν το μέγεθος των μεσαίων συνελικτικών επιπέδων και ρύθμισαν το άλμα (stride) και το μέγεθος του φίλτρου του πρώτου επιπέδου να είναι μικρότερο. Η αρχιτεκτονική του μοντέλου διαφαίνεται στο παρακάτω σχήμα:

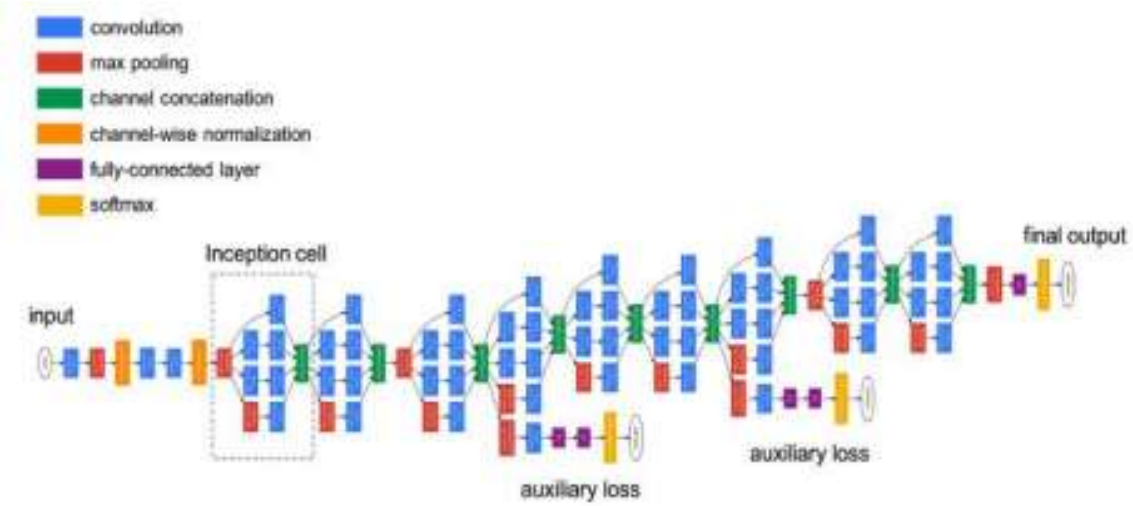


Σχήμα 23: Σχηματική απεικόνιση του μοντέλου ZFNet

1.7.4GoogLeNet (InceptionV1)

Νικητής με αυτό το μοντέλο, στο διαγωνισμό LSVRC του ImageNet, ήταν ο Szegedy [4] από τη Google το 2014. Η βασική του συνεισφορά ήταν η ανάπτυξη του λεγόμενου “Inception Module “ το οποίο μείωσε δραματικά τον αριθμό των παραμέτρων του δικτύου σε 4 εκατομμύρια από 60 που είχε το AlexNet. Επιπρόσθετα, σε αυτό το άρθρο χρησιμοποιείται μέση συγκέντρωση (average pooling) αντί για πλήρως συνδεδεμένα επίπεδα στην κορυφή του ΣΝΔ, εξαλείφοντας μεγάλο πλήθος παραμέτρων οι οποίες δε δείχνουν τελικά να έχουν και τόσο μεγάλη σημασία.

Στο δίκτυο, προκειμένου να βελτιωθεί η συνολική του απόδοση, προστέθηκαν δύο βοηθητικές έξοδοι κατά μήκος του, αυτό θα φανεί καλύτερα στο σχήμα που ακολουθεί παρακάτω. Αργότερα ανακαλύφθηκε, βέβαια, ότι η πρώτη βοηθητική έξοδος δεν προσέδιδε ιδιαίτερη διαφορά στην ποιότητα του τελικού αποτελέσματος του δικτύου. Αυτή η προσθήκη των βοηθητικών εξόδων αρχικά ωφέλησε τις τελικές επιδόσεις του μοντέλου, συγκλίνοντας σε μια ελαφρώς καλύτερη τιμή σε σχέση με το ίδιο δίκτυο που δεν χρησιμοποιούσε όμως τις βοηθητικές εξόδους. Πιστεύεται ότι αυτή η προσθήκη προσδίδει ιδιότητες κανονικοποίησης στο δίκτυο. Το σχήμα αυτής της αρχιτεκτονικής είναι το εξής:



Σχήμα 24: Σχηματική απεικόνιση του μοντέλου GoogLeNet (InceptionV1)

1.7.5VGGNet

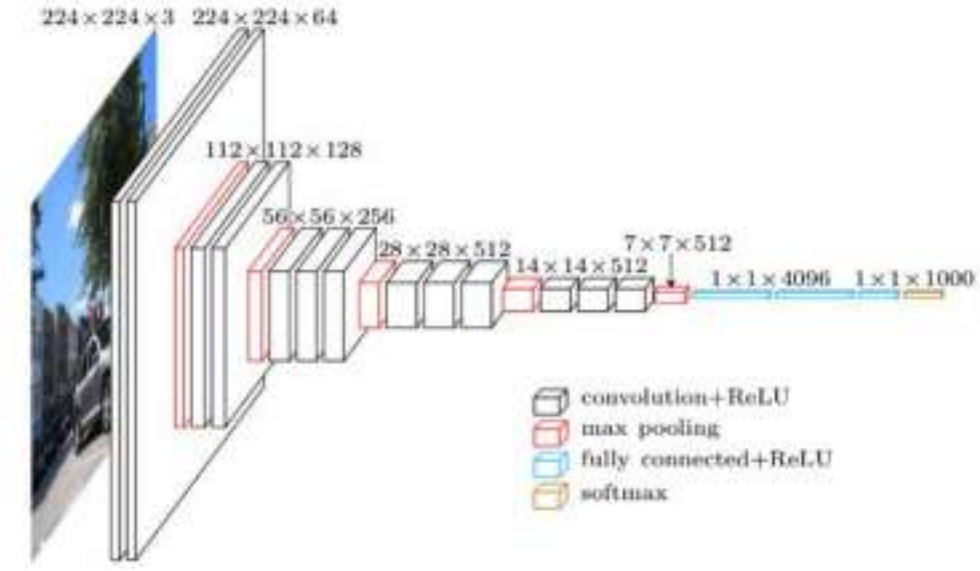
Στο διαγωνισμό LSVRC του ImageNet του 2014, αμέσως μετά τον Szegedy, ήταν οι Karen Simonyan και Andrew Zisserman, με το μοντέλο που έγινε γνωστό ως VGGNet [5]. Η βασική του συνεισφορά είχε να κάνει με το ότι απέδειξε πως το βάθος του δικτύου είναι ένα κρίσιμο συστατικό στοιχείο για καλή απόδοση. Το τελικό και καλύτερό τους μοντέλο περιέχει 16 Συνελικτικά/Πλήρη επίπεδα, και χαρακτηρίζει μία εξαιρετικά ομογενή αρχιτεκτονική που εκτελεί μόνο 3x3 συνελίξεις και 2x2 συγκεντρώσεις από την αρχή μέχρι το τέλος.

Το μειονέκτημα αυτού του μοντέλου βρίσκεται στο ότι είναι πιο κοστοβόρο στην αξιολόγηση καιχρησιμοποιεί περισσότερη μνήμη και παραμέτρους (138 εκατομμύρια). Οι περισσότερες αυτές τις παραμέτρους βρίσκονται στο πρώτο πλήρες συνδεδεμένο επίπεδο, ευτυχώς όμως αργότερα αποδείχτηκε οτι η αφαίρεση τέτοιων επιπέδων δεν υποβαθμίζει την απόδοση του δικτύου με αποτέλεσμα την μεγάλη μείωση του πλήθους των παραμέτρων. Η αρχιτεκτονική αυτού του μοντέλου μπορεί να φανεί στο παρακάτω σχήμα:

Σχήμα 25: Σχηματική απεικόνιση του μοντέλου VGGNet

1.7.6ResNet

Ο Kaiming He, νίκησε στο διαγωνισμό ILSVRC το 2015, με το Residual Network που ανέπτυξε [6]. Το μοντέλο αυτό χαρακτηρίζεται από ειδικά skip connections και βαριά χρήση του batch normalization [7] καθώς επίσης επιτρέπεται μέσω αυτής της αρχιτεκτονικής, η ανάπτυξη πολύ βαθύτερων δικτύων, με εκατοντάδες επίπεδα σε σχέση με τα μέχρι

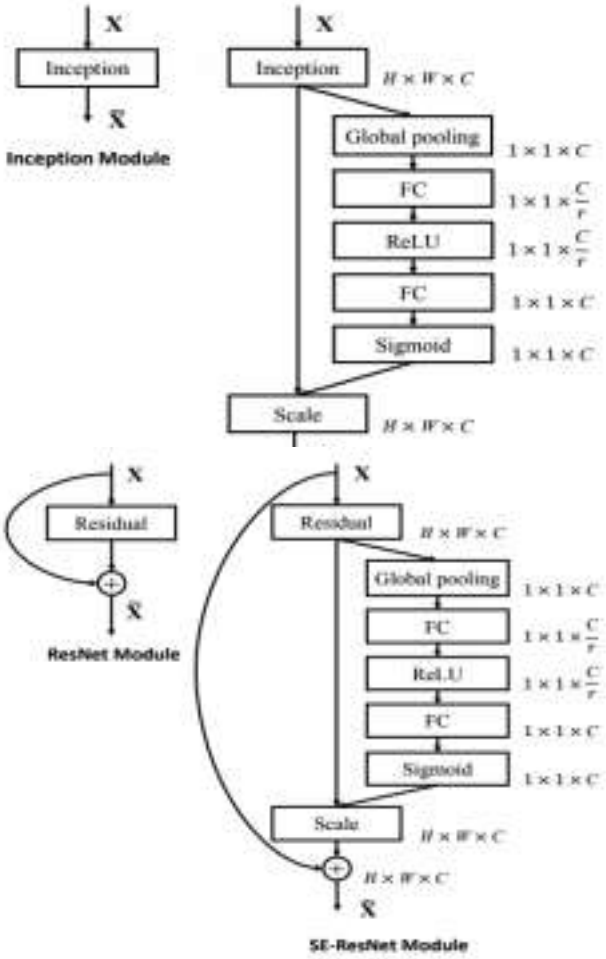


πρότινος δίκτυα που είχαν κάποιες δεκάδες επίπεδα. Από την αρχιτεκτονική λείπουν τα πλήρως συνδεδεμένα επίπεδα στο τέλος του δικτύου και διαθέτει 25 εκατομμύρια παραμέτρους. Το ResNet είναι η εξ ορισμού υλοποίηση των ΣΝΔ στην πράξη από το Μάιο του 2016. Η αρχιτεκτονική τους φαίνεται στο παρακάτω σχήμα:

Σχήμα 26: Σχηματική απεικόνιση του μοντέλου ResNet

1.7.7SENet

Στο τέλος του 2017 παρουσιάστηκε ένα σύνολο δεδομένων μεγάλης κλίμακας με πρόσωπα με το όνομα VGGface2 και το οποίο περιέχει μεγάλη ποικιλία σε πόζες, ηλικίες, φωτισμό, εθνικότητες, και επαγγέλματα. Ο Cao εκπαίδευσε το μοντέλο του SENet [8] με το σύνολο δεδομένων MS-celeb-1M, και μετά το συντόνισε σωστά με το VGGface2 που αναφέρθηκε προηγουμένως, νικώντας στο διαγωνισμό ILSVRC το 2017. Πιο συγκεκριμένα, αυτό που εισάγεται στο μοντέλο αυτό είναι το “Squeeze-and-Excitation block” , μία αρχιτεκτονική μονάδα που σχεδιάστηκε για να βελτιώνει την αντιπροσωπευτική ισχύ ενός δικτύου, επιτρέποντάς του να εκτελέσει επαναβαθμονόμηση χαρακτηριστικών κανάλι-προς-κανάλι (channel-wise feature recalibration). Μέσα από πολλά πειράματα αποδείχτηκε η αποτελεσματικότητα των SENets, ή οποία αγγίζει τις βέλτιστες αποδόσεις (state-of-the-art performance) για πολλά και διάφορα σύνολα δεδομένων και για δίαφορες εργασίες πάνω σε αυτά.



Σχήμα 27: Ένα αυθεντικό Inception (GoogLeNet) στα αριστερά και ενα SE-Inception Module στα δεξιά

Σχήμα 28: Ένα αυθεντικό Residual module (ResNet) στα αριστερά και ενα SE-ResNet Module στα δεξιά

1.8 Συναρτήσεις Απώλειας (Loss Functions) - Γενικά

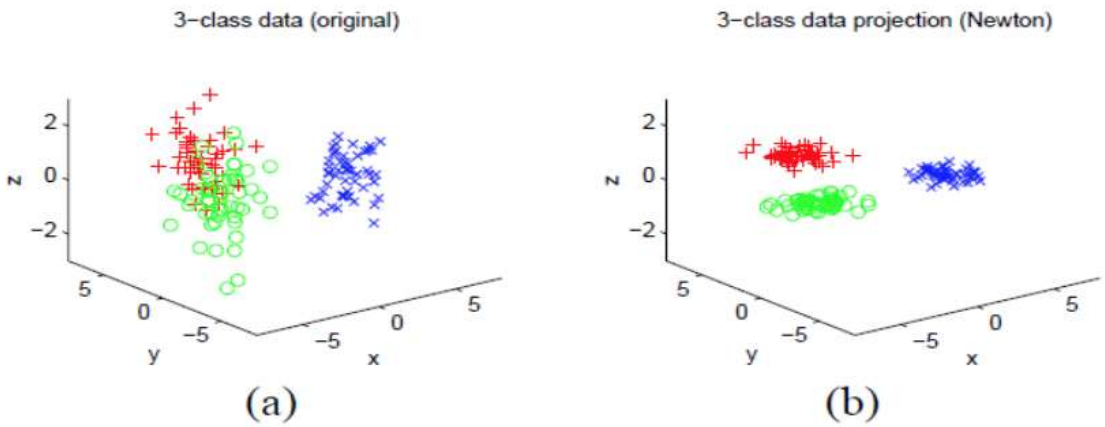
Η συνάρτηση απώλειας είναι ένα μέτρο που χρησιμοποιούμε για να συγκρίνουμε την πρόβλεψη ενός μοντέλου με το αναμενόμενο αποτέλεσμα. Όσο μειώνεται η τιμής της συνάρτησης τόσο αυξάνεται η ευρωστία του μοντέλου και η τιμή του μέτρου είναι πάντα θετική. Οι διακριτικές συναρτήσεις απώλειας (Discriminative Loss Functions) έχουν στόχο την αποδοτική γενίκευση των χαρακτηριστικών , πιο συγκεκριμένα να έχουν την δυνατότητα να διαχωρίζουν αποδοτικά σε διαφορετικές κλάσεις και διαφορετικά σύνολα χαρακτηριστικών μέσα από μεγάλα σύνολα δεδομένων.

Παρακάτω θα αναλύσουμε τις σημαντικότερες και περισσότερο χρησιμοποιούμενες διακριτικές συναρτήσεις απώλειας.

1.8.1 Συνάρτηση απώλειας βασισμένη στην Ευκλείδεια απόσταση

Η συγκεκριμένη συνάρτηση είναι μια μετρική μέθοδος εκμάθησης που ακολουθεί την συγκεκριμένη μεθοδολογία :

- Κάνει ανακατασκευή των δεδομένων x σε $A^{1/2}x$ στον Ευκλείδειο χώρο
- Προσπαθεί να μετακινήσει τα όμοιο ζεύγη πιο κοντά
- Ομαδοποιεί τα όμοια ζεύγη και ταυτόχρονα κρατάει μακριά τα ανόμοια ζεύγη



Σχήμα 29: (α) Αρχικά δεδομένα (β) Ανακατασκευασμένα Δεδομένα (οι κλάσεις είναι πια διαχωρίσιμες)

1.8.2 Συγκριτική συνάρτηση απώλειας (Contrastive Loss Function)

Η συγκεκριμένη συνάρτηση είναι μια από τις πιο χρησιμοποιούμενες συναρτήσεις απώλειας η οποία ακολουθεί την συγκεκριμένη μεθοδολογία :

- Ομαδοποιεί τα ζεύγη εικόνων με πρόσωπα του ίδιου ατόμου
- Απομακρύνει τα ζεύγη εικόνων με πρόσωπα απο διαφορετικά άτομαβαισμένη στην L2-Norm

Όπου $\mathbf{f}_i$και $\mathbf{f}_j$είναι DeepID2 διανύσματα τα οποία έχουν αποσπασθεί από τις εικόνες των δύο συγκρινόμενων δεδομένων.

- Ο πάνω κλάδος της συνάρτησης αναφέρεται στο ζεύγος ίδιων ταυτοτήτων και μειώνει την απόσταση των διανυσμάτων.
- Ο κάτω κλάδος της συνάρτησης αναφέρεται στο ζεύγος διαφορετικών ταυτοτήτων και αυξάνει την απόσταση(των διανυσμάτων fi και fj) η οποία πρέπει να είναι μεγαλύτερη από ένα περιθώριο m (Θue = {m}).

1.8.3Συνάρτηση απώλειας τριπλέτας (Triplet Loss Function)

Η συνάρτηση απώλειας τριπλέτας υπολογίζει τη σχετική απόσταση μεταξύ των ζευγαριών όμοιων εικόνων και ανόμοιων αντίστοιχα[12][13][14], αντιθέτως με την συγκριτική συνάρτηση απώλειας η οποία στηρίζεται στη σύγκριση των απόλυτων αποστάσεων.

Πιο συγκεκριμένα:

- Έχει τη λογική μιας μετρικής εκμάθησης
- Χρησιμοποιείται για να δημιουργήσει διανύσματα χαρακτηριστικών
- Οι προβολές των διανυσμάτων είναι τόσο διακριτές όσο και συμπαγείς
- Πετυχαίνουν μείωση των διαστάσεων την ίδια στιγμή
- Πετυχαίνουν καλύτεροι χρόνοι κατά την εκπαίδευση
- Πλεονεκτήματα αναφορικά με θέματα μνήμης
- Μετα-επεξεργαστικές λειτουργίες όπως. Είναι ο κατακερματισμός και η εικονοποίηση των αποτελεσμάτων.

Σύμφωνα με το [12] :

$$(\mathrm{W}_a)^T \cdot (\mathrm{W}_p) > (\mathrm{W}_a)^T \cdot (\mathrm{W}_n)$$

Έχοντας ως μία τριπλέτα τα {α, n, p} με το α να είναι η άγκυρα (anchor) εικόνα , το p ένα θετικό (positive) παράδειγμα που όμως p ≠ α, και τέλος, n είναι το αρνητικό (negative) παράδειγμα, στόχος μας είναι να βρούμε μια γραμμική προβολή W των δεδομένων ώστε να ικανοποιείται ο παραπάνω περιορισμός που πρακτικά λέει στον χώρο μικρότερων διαστάσεων W θα πρέπει η ομοιότητα μεταξύ α, p να είναι μεγαλύτερη από την ομοιότητα n, p.

Άρα για σύνολο δεδομένων που έχει δοθεί θα πρεπει πρέπει να λυθεί το παρακάτω πρόβλημα βελτιστοποίησης:

$$\underset{\mathbf{W}}{\operatorname{argmin}} \sum_{a,p,n\in\mathbb{T}} max(0, \alpha + a^T \mathbf{W}^\mathbf{T}\mathbf{W}n - a^T \mathbf{W}^\mathbf{T}\mathbf{W}p)$$

Όπου T ένα σύνολο με τριπλέτες , α μία παράμετρος περιθωρίου επιλεγμένο με βάση το σύνολο πιστοποίησης, και για να επιτευχθεί η λύση της προαναφερθείσας σχέσης με τον **Αλγόριθμο Απότομης Καθόδου (Stochastic Gradient Descent)**, το βήμα ενημέρωσης είναι:

$$W_{t+1} = W_t - \eta * W_t * (a(n - p)^T + (n - p)a^T)$$

Η αστάθεια που μπορεί να προκύψει κάποιες φορές λόγω των μεγάλων όγκων δεδομένων που χρησιμοποιούνται για την εκπαίδευση είναι ένα από τα αρνητικά αυτής της συνάρτησης, όπως επίσης είναι και η αργή σύγκλιση. Η προσεκτική επιλογή ζευγαριών εικόνων ή τριπλετών μπορεί να απαλύνει μερικώς αυτό το πρόβλημα.

1.8.4Συνάρτηση απώλειας κέντρου (Center Loss Function)

Η συνάρτηση απώλειας κέντρου μειώνει την απόσταση κάθε σημείου δεδομένων προς το κέντρο των βαθιών χαρακτηριστικών της κλάσης του και προσδίδει ενα βάρος στις αποστάσεις αυτών των χαρακτηριστικών από τα αντίστοιχα κέντρα τους.

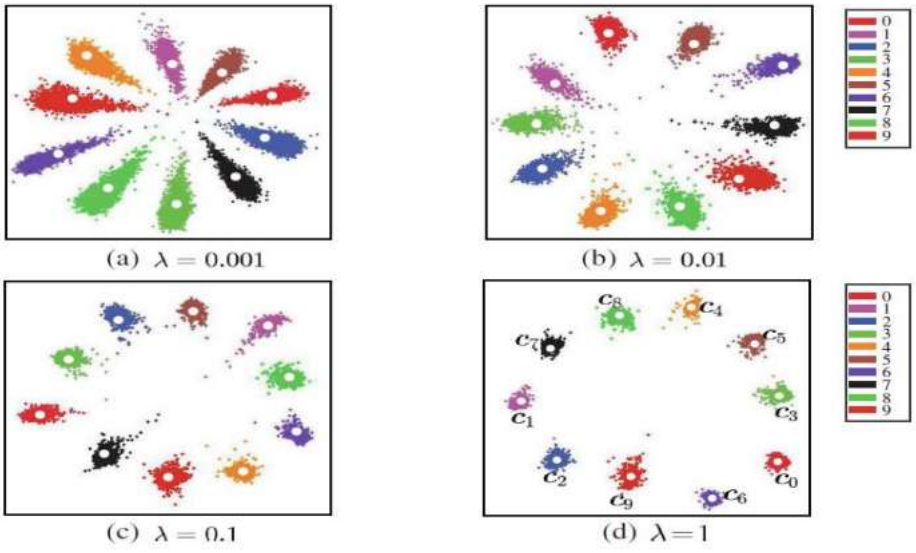
Πιο συγκεκριμένα :

- Η εκπαίδευση δεν είναι τόσο δύσκολη όσο είναι στη Συνάρτηση απώλειας τριπλέτας
- Η συγκεκριμένη συνάρτηση είναι πάρα πολύ καλή για να χρησιμοποιηθεί σε συνδυασμό με τησυνάρτηση απώλειας softmax και να εκπαιδεύσουν ένα εύρωστο Συνελικτικό Νευρωνικό Δίκτυο το οποίο να μπορεί να δώσει βαθιά χαρακτηριστικά με τις εξής πολύ σημαντικές ιδιότητες για το πρόβλημα της αναγνώρισης δεδομένων και να εμποδίζει την κατάρρευση των ενσωματωμένων στοιχείων.

Ο ορισμός της συνάρτησης απώλειας κέντρου παρουσιάζεται παρακάτω :

$$\mathcal{L}_C = \frac{1}{2}\sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2$$

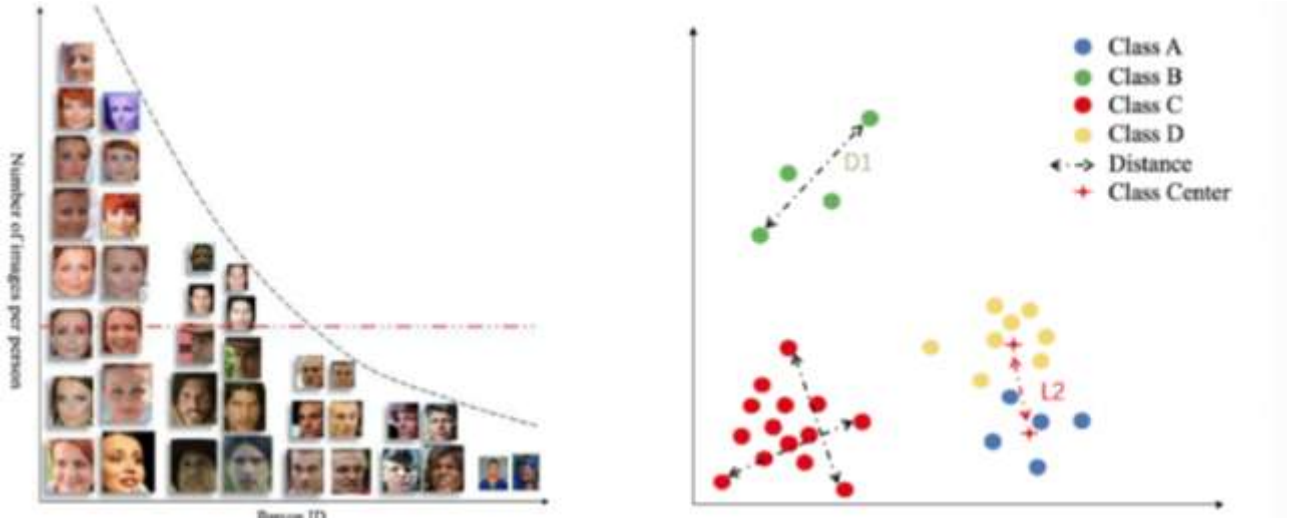
Όπου C_{yi}το κέντρο της κάθε y κλάσης



Σχήμα 30:Softmax Loss και Center Loss. Τα διαφορετικά χρώματα υποδηλώνουν χαρακτηριστικάαπο διαφορετικές κλάσεις.Όπου κάθε λ οδηγεί σε διαφορετική κατανομή c

1.8.5Συνάρτηση απώλειας Εύρους (Range Loss Function)

Η συγκεκριμένη συνάρτηση έχει ως στόχο να δημιουργεί έντονα διακριτές κλάσεις και συμπαγείς, στις περιπτώσεις εντελώς “ανισόρροπων” δεδομένων που ακολουθούν την κατανομή της Μακριάς Ουράς (Long Tailed Distribution) [16]. Η κατανομή αυτή ουσιαστικά αναφέρεται σε κλάσεις που τα δείγματα της είναι πάρα πολλά. Κατά την εκπαίδευση τα αποτελέσματα δε θα είναι ιδιαίτερα καλά λόγω των κλάσεων με λίγα δείγματα τα οποία συνηθίζεται να κόβονται από το σύνολο δεδομένων.



Η προαναφερόμενη συνάρτηση απώλειας υπολογίζει τον αρμονικό μέσο των k μεγαλύτερων αποστάσεων των στοιχείων των κλάσεων, δηλαδή D1 για την κλάση B μαζί με τα αντίστοιχα μέγιστα μήκη αποστάσεων μέσα στις άλλες κλάσεις, και την μικρότερη απόσταση μεταξύ διαφορετικών κλάσεων, δηλαδή L2 για τα κέντρα των κλάσεων A, D .

1.8.6Συνάρτηση απώλειας Softmax

Η συνάρτηση αυτή είναι ο λογαριθμικός κανονικοποιητής της γενικευμένης κατανομής Μπερνουλί και χρησιμοποιείται έτσι ώστε το άθροισμα όλων των πιθανοτήτων να είναι 1. Η εφαρμογή της μας δίνει τις προβλεπόμενες πιθανότητες για κάθε κλάση δεδομένου ενός διανύσματος x:

$$P(y=j|\mathbf{x}) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$$

Είναι η πιο διαδεδομένη συνάρτηση απώλειας για κατηγοριοποίηση και παρουσιάζεται από την ακόλουθη σχέση:

$$L_1 = -\frac{1}{m}\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}},$$

όπου το xi συμβολίζει τα βαθιά χαρακτηριστικά του ι-οστού δείγματος, που ανήκει στην y-οστή κλάση. Η διάσταση χαρακτηριστικών έχει οριστεί σε d = 512. Το Wj συμβολίζει τη j-οστή στήλη των βαρών W στο τελευταίο πλήρως συνδεδεμένο επίπεδο και το b ειναι ο όρος μεροληψίας (bias). Το m και το n είναι αντίστοιχα το μέγεθος του συνόλου εκπαίδευσης και το πλήθος των κλάσεων. Το αρνητικό της softmax συνάρτησης είναι ότι ενώ είναι καλή για την διαφοροποίηση των κλάσεων, χωλαίνει στο πόσο συμπαγή μπορεί να κάνει μία κλάση πράγμα το οποίο οδηγεί σε διαχωρίσιμες κλάσεις αλλά όχι ευδιάκριτες (separable but not discriminative).

1.8.7Συνάρτηση απώλειας L – Softmax

Η συνάρτηση απώλειας L-Softmax είναι μια γενικευμένη συνάρτηση απώλειας Softmax με μεγάλη ενδοκλασική συμπαγή και διαχωριστική ικανότητα μεταξύ των εκπαιδεύσεων.

Με λίγα λόγια είναι μια πιο αυστηρή εκδοχή της Softmax οπου η σχέση της παρουσιάζεται παρακάτω:

$$\|W_1\|\|x\|\cos(\theta_1) \geq \|W_1\|\|x\|\cos(m\theta_1)$$
$$> \|W_2\|\|x\|\cos(\theta_2).$$

Ο μεσαίος όρος της διπλής ανισότητας είναι αυτος που κάνει πιο αυστηρή την κατηγοριοποίηση. Χωρίς αυτο το μέλος της ανίσωσης ο ορισμός είναι ο ίδιος με αυτήν της αυθεντικής Softmax. Ο ορισμός της συνάρτησης δίνεται απο τον παρακάτω τύπο:

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}}{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

$$\text{όπου:} \qquad \psi(\theta) = \begin{cases} \cos(m\theta), & 0 \leq \theta \leq \dfrac{\pi}{m} \\ \mathcal{D}(\theta), & \dfrac{\pi}{m} < \theta \leq \pi \end{cases}$$

1.8.8Συνάρτηση απώλειας A-Softmax

Βασιζόμενη στην συνάρτηση απώλειας L-Softmax, κανονικοποιεί τα βάρη W με βάση την L2-Norm, με αποτέλεσμα τα κανονικοποιημένα διανύσματα να βρίσκονται επάνω σε μια υπερσφαίρα, με αποτέλεσμα τα διακριτά χαρακτηριστικά των δεδομένων να μαθαίνονται σε μια επιφάνεια υπερσφαίρας με βάση ένα γωνιακό περιθώριο.

Παρουσιάζεται από την παρακάτω σχέση:

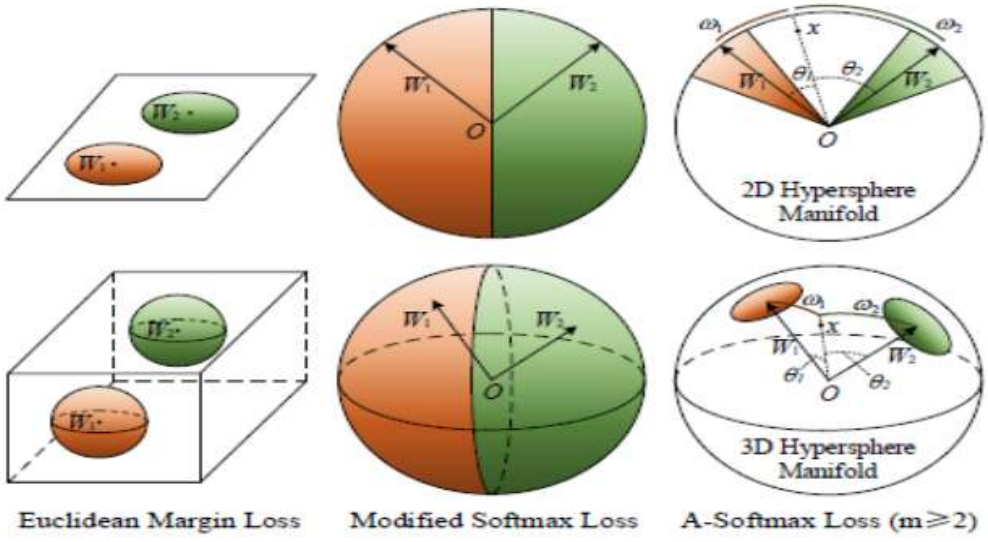
$$L = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{e^{\|x_i\|\psi(\theta_{y_i})}}{e^{\|x_i\|\psi(\theta_{y_i})} + \sum_{j=1,j\neq y_i}^{n} e^{\|x_i\|\cos\theta_j}},$$

$$\text{όπου:} \quad \psi(\theta_{y_i}) = (-1)^k\cos(m\theta_{y_i}) - 2k, \theta_{y_i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right], k \in [0, m-1], \; m \geqslant 1$$

Με το m να είναι ένας θετικός ακέραιος που ελέγχει το μέγεθος του περιθωρίου και με την ψ(θ) η όποια είναι μία συνάρτηση που θα προσδώσει μονοτονία στην συνάρτηση L, διότι από μόνη της η συνημοτονοειδής συνάρτηση δεν είναι μονότονη. Πρέπει να αναφερθεί πως είναι απαραίτητη η χρήση της Softmax για επίβλεψη για να εξασφαλιστεί σύγκλιση κατά την εκπαίδευση, και το τα βάρη να ελέγχονται από μια δυναμική παράμετρο λ. Συνεπώς με την πρόσθετη συνάρτηση απώλειας softmax η ψ(θ) πλέον γίνεται:

$$\psi(\theta_{y_i}) = \frac{(-1)^k \cos(m\theta_{y_i}) - 2k + \lambda \cos(\theta_{y_i})}{1 + \lambda}$$

Είναι σημαντικό να δίνεται ιδιαίτερη προσοχή στην επιλογή του λ διότι η ύπαρξή του μπορεί να να κρύβει παγίδες ως προς την εκπαίδευση του συστήματος.



Σχήμα 32:Γεωμετρική ερμηνεία συνάρτησης απώλειας για Ευκλείδεια περιθώρια

Στο σχήμα 32 Απεικονίζεται η γεωμετρική ερμηνεία συνάρτησης απώλειας για Ευκλείδεια περιθώρια όπως είναι οι: Contrastive Loss, Triplet Loss, Center Loss, Range Loss κτλ, τροποποιημένη Softmax Loss και τέλος A-Softmax Loss. Όπου πορτοκαλί και πράσινο οι περιορισμοί για διακρισιμότητα των κλάσεων 1 και 2 αντίστοιχα.

Το περιθώριο απόφασης αυτής της συνάρτησης εξαρτάται από τη γωνία θ η οποία οδηγεί σε διαφορετικά περιθώρια για διαφορετικές κλάσεις. Αυτό έχει σαν αποτέλεσμα στο χώρο αποφάσεων μερικά χαρακτηριστικά διαφορετικών κλάσεων, να έχουν μεγαλύτερο περιθώριο ενώ άλλα να έχουν μικρότερο, πράγμα το οποίο μειώνει την δυνατότητα διακρισιμότητας μεταξύ των κλάσεων (discriminating power reduction).

1.8.9Συνάρτηση απώλειας Additive Cosine Margin

Είναι μία συνάρτηση η οποία εισάγει το συνημοτονειδές περιθώριο στην συνάρτηση της softmax μέσω του cosθ – m το οποίο είναι απλούστερο στην υλοποίηση από την υπερ-παράμετρο λ [18][19] η οποία χρειάζεται για να εξασφαλίσει έλεγχο του βάρους. Επιπλέον πλεονεκτήματα αυτής της συνάρτησης απώλειας είναι ότι μπορεί να συγκλίνει χωρίς την επίβλεψη της Softmax, και ότι παρουσιάζει καλύτερες επιδόσεις σε σχέση με όλες τις προαναφερθείσες συναρτήσεις απώλειας.

Συνεπώς με την απώλεια της Softmax ο τύπος παρουσιάζεται απο την ακόλουθη σχέση:

$$L = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{e^{s(\cos(\theta_{y_i})-m)}}{e^{s(\cos(\theta_{y_i})-m)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cos\theta_j}}$$

1.8.10Συνάρτηση απώλειας Additive Cosine Margin

Παρά τις ομοιότητες της με την Additive Cosine Margin,η Additive Angular Margin έχει μια πιο καθαρή γεωμετρική ερμηνεία καθώς απευθύνεται στην απόσταση του τόξου στην επιφάνεια της υπερσφαίρας,και δίνεται από την παρακάτω σχέση:

$$L = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m))}+\sum_{j=1,j\neq y_i}^{n}e^{s\cos\theta_j}}$$

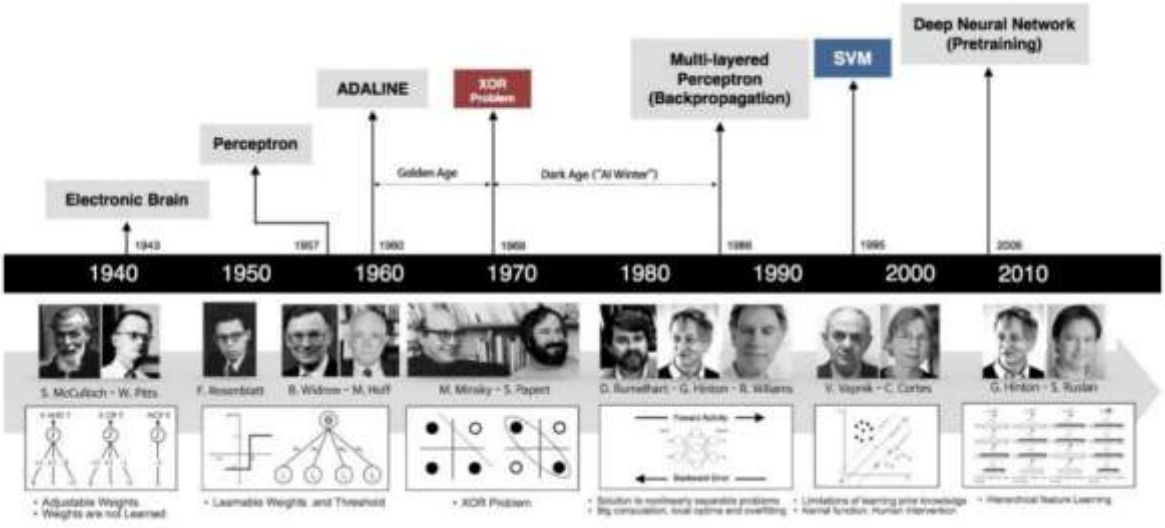
Όπου σε αντίθεση με την προηγούμενη συνάρτηση απώλειας, βλέπουμε εσωτερική αύξηση στο συνημίτονο (δηλ. γωνιακή αύξηση) κι επειδή το cos(θ+m) είναι μικρότερο απο το cos(θ) όταν $\theta \in [0 , \pi - m]$ ο περιορισμός είναι πιο αυστηρός για κατηγοριοποίηση, [21][22].pergamos references

1.9 Βαθιά Μάθηση (Deep Learning) - Εισαγωγή

Η βαθιά μάθηση (επίσης γνωστή ως βαθιά δομημένη μάθηση ή διαφορικός προγραμματισμός) είναι μέρος μιας ευρύτερης οικογένειας μεθόδων της μηχανικής μάθησης που βασίζεται σε τεχνητά νευρωνικά δίκτυα με μαθησιακή αναπαράσταση. Η μάθηση αυτή μπορεί να εποπτεύεται, να ημι-εποπτεύεται ή να μην παρακολουθείται.

1.9.1Ιστορική αναδρομή

Μέχρι πρόσφατα, οι περισσότερες τεχνικές μηχανικής μάθησης και επεξεργασίας σήματος εκμεταλλεύονταν ρηχές (shallow-structured) αρχιτεκτονικές. Αυτές οι αρχιτεκτονικές τυπικά περιέχουν το πολύ ένα ή δύο στρώματα μετασχηματισμών μη γραμμικών χαρακτηριστικών. Χαρακτηριστικά παραδείγματα αποτελούν τα μοντέλα Λογιστικής Παλινδρόμησης, Μέγιστης Εντροπίας (MaxEnt), Support Vector Machine (SVM) και πολυεπίπεδα Perceptrons (MLP) με ένα κρυμμένο στρώμα. Αυτές οι ρηχές αρχιτεκτονικές αποδείχθηκαν αρκετά αποτελεσματικές σε πολλά απλά και καλά δομημένα προβλήματα, αλλά λόγω της περιορισμένης ισχύς μοντελοποίησης τους, παρουσιάζουν αρκετά μεγάλη δυσκολία σε πιο σύνθετα καθημερινά προβλήματα. Η ανάπτυξη των τεχνητών νευρωνικών δικτύων συνδέεται στενά με την ανάπτυξη στις επιστήμες της Βιολογίας και της Νευρολογίας. Έτσι, οι πρώτες ενδείξεις για την ανάγκη επεξεργασίας της πληροφορίας σε πολλαπλά επίπεδα, προήλθε από τις επιστήμες αυτές, όπου έχει παρατηρηθεί ότι το ανθρώπινο νευρικό σύστημα, αποτελείται από πολλαπλά επίπεδα επεξεργασίας

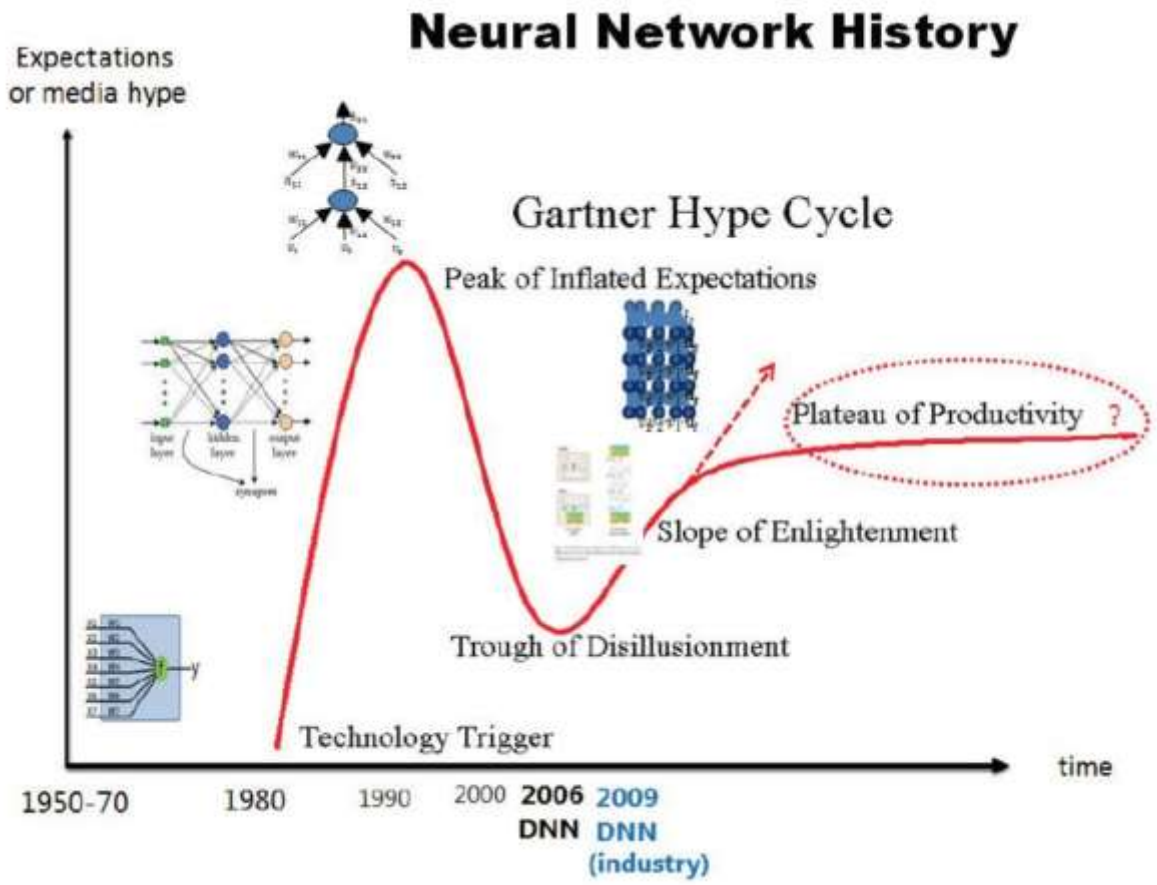


Σχήμα 33:Χρονολογίες ορόσημα στην ανάπτυξη νευρωνικών δικτύων
By Andrew L. Beam - machine learning and medicine.

Η ιδέα της δημιουργίας μιας «μηχανής σκέψης» ξεκινάει το 1950 με τον Alan Turing, ο οποίος στο βασικό του έγγραφο «Υπολογιστικές Μηχανές και Νοημοσύνη» έθεσε διάφορα κριτήρια για να κρίνει αν μια μηχανή θα μπορούσε να ειπωθεί ως έξυπνη, η οποία από τότε έγινε γνωστή ως «δοκιμή Turing». Η ιστορία της Βαθιάς Μάθησης όμως, ουσιαστικά ξεκινάει από το 1943, όταν ο Walter Pitts και ο Warren McCulloch δημιούργησαν ένα μοντέλο υπολογιστή βασισμένο στα νευρωνικά δίκτυα του ανθρώπινου εγκεφάλου. Χρησιμοποίησαν έναν συνδυασμό αλγορίθμων και μαθηματικών που ονόμαζαν «λογική κατωφλίου» για να μιμηθούν τη διαδικασία σκέψης. Δεν είναι όμως, μέχρι το "perceptron"του Frank Rosenblatt το 1959 όπου βλέπουμε τον πρώτο πραγματικό πρόδρομο των σύγχρονων νευρωνικών δικτύων. Από τότε, η Βαθιά Μάθηση έχει εξελιχθεί σταθερά, με μόνο δύο σημαντικά διαλείμματα στην ανάπτυξή της. Και οι δύο ήταν συνδεδεμένοι με τους περίφημους «χειμώνες» της τεχνητής νοημοσύνης. (Beam, 2017)

Ο πρώτος «χειμώνας», ήρθε όταν ο Marvin Minsky, ο οποίος συχνά θεωρείται ένας από τους πατέρες της τεχνητής νοημοσύνης, απέδειξε μαζί με τον Seymor Papert ότι το "perceptron"του Rosenblatt δεν ήταν ικανό να μάθει την απλή συνάρτηση XOR, όσο και να το εκπαιδεύαν. Αυτό, τη σημερινή εποχή δεν αποτελεί έκπληξη, καθώς το μοντέλο του perceptron είναι γραμμικό και η συνάρτηση XOR είναι μη γραμμική, αλλά εκείνη την εποχή ήταν αρκετή αυτή η διαπίστωση ώστε να «σκοτώσει» όλη την έρευνα για τα νευρωνικά δίκτυα έχοντας ως αποτέλεσμα τον πρώτο «χειμώνα» της τεχνητής νοημοσύνης.

Στην πορεία το 1980, προτάθηκε από τον Kunihiko Fukushima, ένα ιεραρχικό τεχνητό νευρωνικό δίκτυο, το Neocognitron, το οποίο θεωρείται ο πρόγονος των σημερινών βαθιών νευρωνικών δικτύων. Αν και η δομή του έμοιαζε με των σημερινών, ο τρόπος μάθησης ήταν διαφορετικός και σε συνδυασμό με την μη επαρκή υπολογιστική δύναμη της εποχής, οδήγησε σε μη επιτυχή αποτελέσματα. Το 1986, ο Geoff Hinton μαζί με τους David Rumelhart και Ronald Williams, παρουσίασαν μία εργασία, όπου απέδειξαν ότι τα νευρωνικά δίκτυα με πολλά κρυμμένα στρώματα, θα μπορούν να εκπαιδεύονται αποτελεσματικά με μία σχετικά απλή διαδικασία. Ήταν η μέθοδος Back-propagation, όπου θα επέτρεπε στα νευρωνικά δίκτυα να ξεπεράσουν την αδυναμία του perceptron, αφού τα πρόσθετα στρώματα προμήθευαν το δίκτυο με τη δυνατότητα να μάθουν μη γραμμικές λειτουργίες. Στην πορεία αποδείχθηκε ότι είχαν την ικανότητα να μαθαίνουν οποιαδήποτε συνάρτηση. Αυτό το αποτέλεσμα είναι γνωστό ως θεώρημα γενικής προσέγγισης (universal approximation theorem).



Σχήμα 34:Η πορεία της εξέλιξης των τεχνητών νευρωνικών δικτύων σε σχέση και με τις προσδοκίες στον χρόνο. By Li Deng and Dong Yu - Deep Learning Methods and Applications

Τα επόμενα χρόνια υπήρξε μία περίοδος, όπου λόγω και της υπερβολικής φιλοδοξίας των τότε επιστημόνων, είχε σαν αποτέλεσμα να υπερβάλλουν σε σχέση με το άμεσο δυναμικό (potential) των νευρονικών δικτύων αλλά και της Τεχνητής Νοημοσύνης γενικότερα και σηματοδότησε τον δεύτερο «χειμώνα» της Τεχνητής Νοημοσύνης. Αποτέλεσμα αυτού ήταν ότι η επιστήμη της τεχνητής νοημοσύνης να φθάσει στα όρια της ψευδοεπιστήμης. Ευτυχώς όμως ορισμένοι επιστήμονες συνέχισαν να εργάζονται στον τομέα αυτόν για να ανατρέψουν την κατάσταση και το 1995 η Corinna Cortes και ο Vladimir Vapnik, ανέπτυξαν τα Support Vector Machines (SVM).

Το επόμενο σημαντικό βήμα στην εξέλιξη της Βαθιάς Γνώσης έγινε το 1999 με την ανάπτυξη πλέον ταχύτερων επεξεργαστών και καρτών γραφικών για τους υπολογιστές. Στην περίοδο αυτή τα τεχνητά νευρωνικά δίκτυα ξεκίνησαν να συναγωνίζονται τα SVMs, αφού παρήγαγαν πολύ καλύτερα αποτελέσματα αν και ήταν πολύ πιο αργά σε σχέση με αυτά. (Foote, 2017)

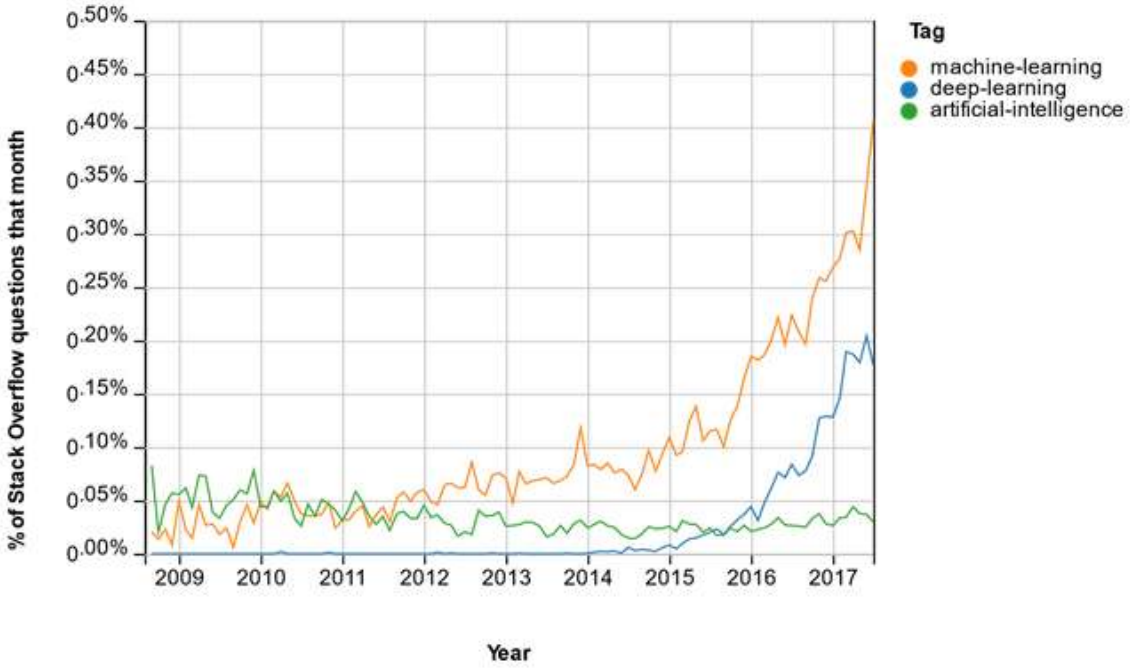
Επί του παρόντος η επεξεργασία των Μεγάλων Δεδομένων (Big Data) και της Τεχνητής Νοημοσύνης, εξαρτώνται από την Βαθιά Μάθηση, η οποία συνεχίζει να εξελίσσεται με ταχύτατους ρυθμούς.

1.9.2 Βαθιά Μάθηση

Τι είναι η Βαθιά Μάθηση; (Deep Learning);

Η Βαθιά Μάθηση είναι ένα λογισμικό που μιμείται το δίκτυο των νευρώνων του εγκαιφάλου.Πρόκειται για ένα υποσύνολο της Μηχανικής Μάθησης και ονομάζεται Βαθιά Μάθηση διότι χρησιμοποιεί βαθιά νευρωνικά δίκτυα.Το μηχάνημα χρησιμοποιεί διαφορετικά επίπεδα για να μάθει απο τα δεδομένα. Το βάθος του μοντέλου αντιπροσωπεύεται απο τον αριθμό των στρωμάτων στο μοντέλο.

Η Βαθιά Μάθηση είναι η νέα κατάσταση της τέχνης απο την άποψη της Τεχνητής Νοημοσύνης.



Σχήμα 35:Οι σχετικές ερωτήσεις στο StackOverflow ανα έτη.By GURU99

Η Βαθιά Μάθηση είναι η πρόοδος στον τομέα της τεχνητής νοημοσύνης. Όταν υπάρχουν αρκετά δεδομένα για την εκπαίδευση, η βαθιά εκμάθηση επιτυγχάνει εντυπωσιακά αποτελέσματα, ειδικά στην αναγνώριση εικόνας και τη μετάφραση κειμένων.

1.9.3Η σχέση με τη Στατιστική

Η εκμάθηση μηχανών και η στατιστική είναι στενά συνδεδεμένα πεδία από άποψη μεθόδων, αλλά πιο συγκεκριμένα στον κύριο στόχο τους.Οι στατιστικές αντλούν τα συμπεράσματα τους από ένα δείγμα, ενώ η μηχανική μάθηση βρίσκει γενικευμένα πρότυπα πρόγνωσης. Σύμφωνα με τον Michael I. Jordan, οι ιδέες της μηχανικής μάθησης, από τις μεθοδολογικές αρχές μέχρι τα θεωρητικά εργαλεία, είχαν μακρά προϊστορία με τη στατιστική. Επίσης πρότεινε τον όρο επιστήμη δεδομένων (Data Sience) ως ονομασία του κλάδου.

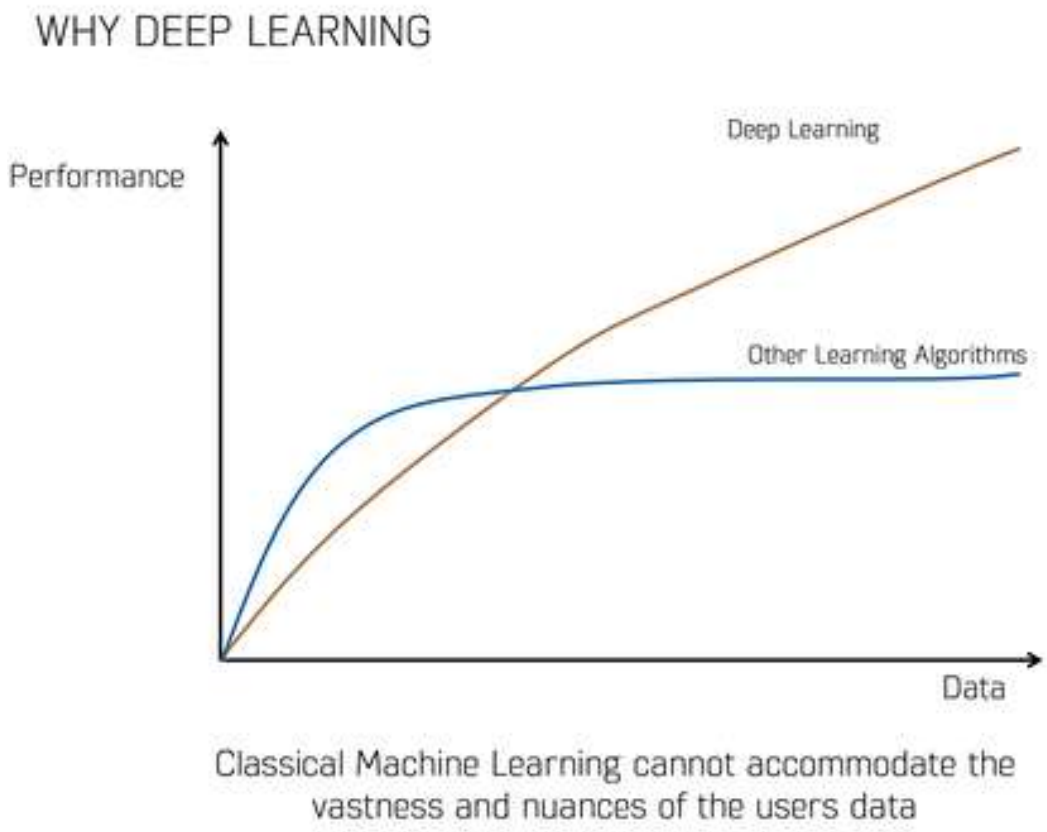
Ο Leo Breiman διακρίνει δύο παραδείγματα στατιστικής μοντελοποίησης: το μοντέλο δεδομένων και το αλγοριθμικό μοντέλο, όπου "αλγοριθμικό μοντέλο” εννοεί τους αλγόριθμους μηχανικής μάθησης όπως το τυχαίο δάσος.

Μερικοί στατιστικολόγοι υιοθέτησαν μεθόδους από τη μηχανική μάθηση, οδηγώντας σε ένα συνδυασμένο πεδίο που ονομάζουν στατιστική μάθηση (Statistical Learning).

1.9.4Γιατί Βαθιά Μάθηση;

Η βασική κατανόηση του ανθρώπινου εγκεφάλου, της λειτουργίες του και οι λύσεις πρακτικών προβλημάτων της καθημερινότητας μας είναι μερικοί από τους λόγους για να ασχοληθούμε με την Βαθιά Μάθηση και πιο συγκεκριμένα με τα νευρωνικά δίκτυα.Μεγάλο πλεονέκτημα είναι ότι οι αλγόριθμοι μπορούν συνεχώς να εκπαιδεύονται με νέα δεδομένα με αποτέλεσμα να βελτιώνονται και να παράγουν καλύτερα αποτελέσματα, με λίγα λόγια αποκτούν εμπειρία.

Έτσι με το πέρασμα του χρόνου η Βαθιά Μάθηση αρχίζει και κερδίζει όλο και περισσότερο έδαφος και αυτό οφείλεται σε δύο τομείς, την υπολογιστική ισχύ και τον μεγάλο όγκο δεδομένων.Έτσι υπερτερούν από πολλές τεχνικές μηχανικής μάθησης όσον αφορά την αποτελεσματικότητα και την αποδοτικότητα τους σε όλο και περισσότερες εφαρμογές, λόγω και της χρήσης πολλαπλών στρωμάτων στην επεξεργασία των δεδομένων. Τέλος, μπορούμε να πούμε ότι οι αλγόριθμοι βαθιάς μάθησης είναι πιο



εύχρηστοι από πολλούς αλγόριθμους μηχανικής μάθησης. (Woodie, 2017)

By Andrew NG

1.9.5 Εφαρμογές Βαθιάς Μάθησης

Μερικές από τις πιο σημαντικές εφαρμογές της Βαθιάς Μάθησης είναι οι εξής:

Χρηματοοικονομικά

- Πρόβλεψη συναλλαγματικών μεταβολών
- Διαχείριση χαρτοφυλακίων

- Αξιολόγηση αιτήσεων για δάνειο
- Αποτίμηση ακίνητης περιουσίας

Υγεία

- Ανάλυση συμπτωμάτων και διάγνωση για διάφορες ασθένειες
- Αναγνώριση Καρκίνου του δέρματος
- Ακτινοδιαγνωστική
- Πρόβλεψη ασθένιας απο τα αρχεία ασθενή

Τηλεπικοινωνίες

- Marketing
- Συμπίεση δεδομένων
- Μετάφραση γλώσσας σε πραγματικό χρόνο

Τεχνολογία

- Ρομποτική
- Μεταφορές
- Κατασκευές
- Αεροδιαστημική
- Αυτόνομα Αυτοκίνητα
- Μείωση κατανάλωσης ηλεκτρικού ρεύματος

Στο παρελθόν, η τεχνολογία αυτή συνδεόταν στενά μόνο με τους κεντρικούς υπολογιστές και τους υπερυπολογιστές. Καθώς όμως αναλύεται σε μικρότερα κομμάτια για πιο περιορισμένες εφαρμογές, η Βαθιά Μάθηση θα έχει πολύ μεγαλύτερη επίδραση σε όλο και μεγαλύτερο αριθμό αγορών. Η πορεία αυτής της αγοράς μόλις αρχίζει.

2 Υλοποίηση Προγράμματος

2.1 Dataset

2.1.1 Εισαγωγή

Για την εκπαίδευση ενός νευρωνικού δικτύου είναι απαραίτητο ένα σετ από δεδομένα με τα οποία θα γίνει η εκπαίδευση και η επαλήθευση του δικτύου. Υπάρχουν δύο τρόποι να αποκτήσουμε ένα σετ δεδομένων. Ο πρώτος τρόπος είναι να δημιουργηθούν καινούργιες εικόνες ή να αντλήσουμε εικόνες από το διαδίκτυο οι οποίες στην συνέχεια θα πρέπει να κατηγοριοποιηθούν. Η συγκεκριμένη διαδικασία είναι πολύ χρονοβόρα και συνήθως γίνεται από ομάδες ατόμων ή εθελοντικά από διαφορές κοινότητες. Η πιο ευέλικτη λύση είναι να χρησιμοποιηθεί ένα από τα πολλά έτοιμα σετ δεδομένων τα οποία έχουν ήδη ταξινομηθεί και προεπεξεργαστεί για την τροφοδότηση τους σε ένα νευρωνικό δίκτυο.

2.1.2 Fashion MNIST

Ένα από τα πιο γνωστά σετ δεδομένων είναι το Fashion MNIST το οποίο έχει δημιουργηθεί από προϊόντα της Zalando μια γερμανικής πλατφόρμα αναζήτησης προϊόντων ένδυσης. Το σετ δεδομένων αποτελείτε από 60000 εικόνες εκπαίδευσης καθώς και από 10000 εικόνες επαλήθευσης με τις αντίστοιχες κατηγορίες των εικόνων. Κάθε εικόνα έχει διαστάσεις 28x28 και είναι εικόνα επιπέδου γκρίζου και έχει ταξινομηθεί σε μια από τις 10 διαθέσιμες κλάσεις.

Το σετ δεδομένων δημιουργήθηκες από την εταιρία Zalando για να αντικαταστήσει το κλασικό σετ δεδομένων MNIST το οποίο αποτελείτε από χειρόγραφα ψηφία. Το dataset MNIST χρησιμοποιείτε ευρέως από την κοινότητα της μηχανικής μάθησης και συνήθως οι ερευνητές το χρησιμοποιούν για να ελέγξουν την απόδοση των αλγορίθμων τους. Στην πραγματικά το MNIST είναι το πρώτο σετ δεδομένων που θα τροφοδοτηθεί σε ένα καινούργιο νευρωνικό δίκτυο και λέγεται ότι εάν δεν λειτουργεί το μοντέλο με αυτό το σετ δεδομένων δεν θα λειτουργήσει με κανένα σετ δεδομένων. Στην πραγματικότητα όμως παρόλο που μπορεί να λειτουργήσει στο σετ δεδομένων MNIST υπάρχει μεγαλει

πιθανότητα να μην λειτουργήσει στα υπόλοιπα σετ δεδομένων. Για αυτόν τον λόγο δημιουργήθηκε το σετ δεδομένων Fashion MNIST το οποίο υπόσχεται μεγαλύτερη καληψη περιπτώσεων έτσι ώστε η επαλήθευση των μοντέλων να αντιπροσωπεύει όσο το δυνατών περισσότερο την πραγματικά αποδοτικότητα των αλγορίθμων .
Κάθε εικόνα του σετ δεδομένων αποτελείτε από 28 εικονοστοιχεία στο ύψος και 28 εικονοστοιχεία στο πλάτος που στο σύνολο είναι 784 εικονοστοιχεία. Κάθε εικονοστοιχείο έχει μια ακέραια τιμή η οποία αναπαριστά το πόσο φωτεινό είναι το συγκεκριμένο εικονοστοιχείο. Όσο μικρότερη είναι η τιμή του εικονοστοιχείου τόσο πιο φωτεινό είναι και όσο μεγαλύτερη είναι η τιμή του εικονοστοιχείου τόσο πιο σκοτεινό είναι. Οι τιμές των εικονοστοιχείων ξεκινάνε από το 0 και φτάνουν έως το 255 που είναι το απολυτό μαύρο.

Το σετ δεδομένων κατηγοριοποιεί τις εικόνες σε 10 κλάσεις. Οι διαθέσιμες κλάσεις είναι οι εξής: 0 T-shirt/top ,1 Touser, 2 Pullover, 3 Dress, 4Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot.

Υπάρχει πληθώρα τρόπων για την άντληση του σετ δεδομένων όπως το να κατεβούν από κάποια πλατφόρμα ανοιχτών δεδομένων και να εισαχθούν στο πρόγραμμα. Ο πιο εύκολος και γρήγορος τρόπος είναι να εισαχθούν μέσω της βιβλιοθήκης Keras. Η βιβλιοθήκη δεν θα χρησιμοποιηθεί πέραν της διαδικασίας της εισαγωγής των δεδομένων.

Αρχικά πρέπει να εισαχθεί το τμήμα της βιβλιοθήκης Keras που είναι υπεύθυνο για την εισαγωγή του σετ δεδομένων Fashion MNIST.

Importing the keras library just to import the Fashion MNIST dataset

```
In [3]: 1 from keras.datasets import fashion_mnist
```

Στη συνέχεια θα αντιστοιχήσουμε τις τιμές του σετ δεδομένων στις μεταβλητές που θα χρησιμοποιηθούν από το πρόγραμμα για την πρόσβαση στα δεδομένα.

Loading the Fashion MNIST dataset using the keras library.

```
In [4]: 1 ((X_train, Y_train), (X_test, Y_test)) = fashion_mnist.load_data()
```

Το επόμενο βήμα είναι η εξερεύνηση του σετ δεδομένων για να μελετηθούν τα χαρακτηριστικά του.

Αρχικά θα μελετηθούν οι διαστάσεις των μεταβλητών.

Explore the shapes of our data.

```
In [7]:
1 print(f'The shape of our training images array is: {X_train.shape}')
2 print(f'The shape of our testing images array is: {X_test.shape}')
3 print(f'The shape of our training labels array is: {Y_train.shape}')
4 print(f'The shape of our testing labels array is: {Y_test.shape}')
```

```
The shape of our training images array is: (60000, 28, 28)
The shape of our testing images array is: (10000, 28, 28)
The shape of our training labels array is: (60000,)
The shape of our testing labels array is: (10000,)
```

Όπως φαίνεται στην παραπάνω εικόνα η μεταβλητή X_train έχει διαστάσεις (60000, 28, 28) δηλαδή 60000 εικόνες μεγέθους 28x28 και η μεταβλητή Y_train,που κρατάει τις τιμές που αντιστοιχούν στην κατηγορία της κάθε εικόνας, είναι ένας μονοδιάστατος πίνακας 10000 τιμών. Το σετ δεδομένων αποτελείτε από 10000 εικόνες διαστάσεων 28x28 για αυτό τον λόγο η μεταβλητή X_test είναι διαστάσεων (10000, 28, 28) και η μεταβλητή Y_test είναι ένας μονοδιάστατος πίνακας 10000 θέσεων.

Στη συνέχεια θα μελετήσουμε τα μέγιστα και τα ελάχιστα των δεδομένων του σετ.

Explore the min and the max of our images

```
In [9]:
1 print(f'The max value of our training images array is: {X_train.max()}')
2 print(f'The min value of our training images array is: {X_train.min()}')
3 print(f'The max value of our testing images array is: {X_test.max()}')
4 print(f'The min vlaue of our testing images array is: {X_test.min()}')
```

```
The max value of our training images array is: 255
The min value of our training images array is: 0
The max value of our testing images array is: 255
The min vlaue of our testing images array is: 0
```

Explore the min and the max of our labels

```
In [11]:
1 print(f'The max value of our training labels array is: {Y_train.max()}')
2 print(f'The min value of our training labels array is: {Y_train.min()}')
3 print(f'The max value of our testing labels array is: {Y_test.max()}')
4 print(f'The min vlaue of our testing labels array is: {Y_test.min()}')
```

```
The max value of our training labels array is: 9
The min value of our training labels array is: 0
The max value of our testing labels array is: 9
The min vlaue of our testing labels array is: 0
```

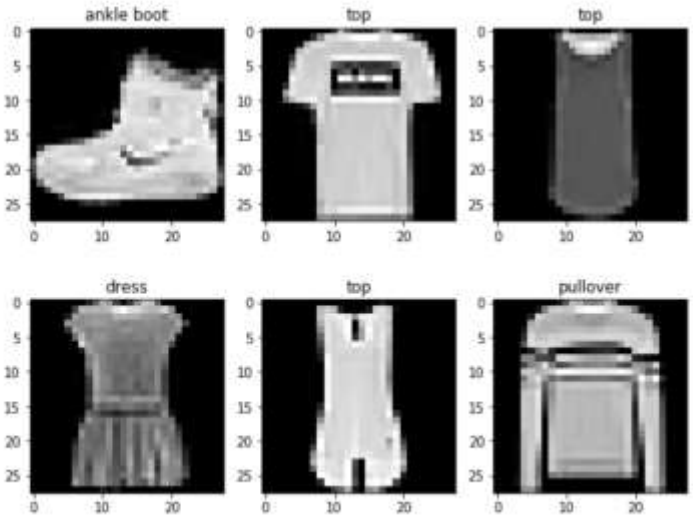
Όπως φαίνεται στην παραπάνω εικόνα οι τιμές των εικόνων κυμαίνονται από 0 έως 255 καθώς οι εικόνες είναι επιπέδου γκρίζου και οι τιμές των κατηγοριών βρίσκονται στο διάστημα (0,9) διότι το σετ αποτελείτε από 10 κατηγορίες.

Στη συνέχεια θα δημιουργηθεί μια λίστα με τους πραγματικούς τίτλους των κατηγοριών έτσι ώστε δίνοντας την τιμή που αντιστοιχεί στην κατηγορία της κάθε εικόνας θα εμφανίζεται ο πραγματικός τίτλος της κατηγορίας.

Στη συνέχεια θα εμφανίσουμε τις έξι πρώτες εικόνες από το σετ δεδομένων.

Display the first six training images with their labels

In [6]:
```
fig1 = plt.figure(figsize=(9,7))
ax1 = []
for i in range(6):
  ax1.append(fig1.add_subplot(2, 3, i+1))
  ax1[-1].set_title(label_names[Y_train[i]])
  plt.imshow(X_train[i], cmap='gray')
plt.show()
```



Όπως φαίνεται παραπάνω η ανάλυση των εικόνων δεν είναι αρκετά καλή για την ανθρώπινη οπτική είναι όμως αρκετή έτσι ώστε να τροφοδοτηθεί στο νευρωνικό δίκτυο και να κατηγοριοποιηθεί από αυτό.

In [5]:
```
label_names = ["top", "trouser", "pullover", "dress", "coat",
               "sandal", "shirt", "sneaker", "bag", "ankle boot"]
```

2.2 Υλοποίηση Μοντέλου

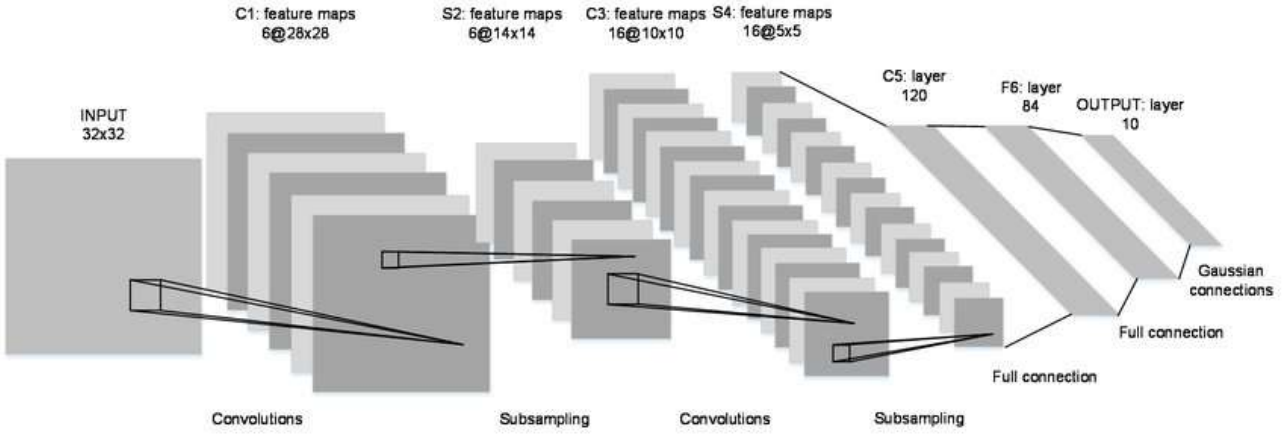
2.2.1 Προεπεξεργασία δεδομένων

Do the necessary image preprocessing

- Reshape the images to include so that the end format is (m, n_H, n_W, n_C).
  Where m is the number of images, n_H is the high of the images, n_W is the width of the images and n_C is the number of channels of each image.
- The LeNet architecture accepts a 32x32 pixel images as input, mnist data is 28x28 pixels. We simply pad the images with zeros to overcome that.
- Normalise the images so that the value of each pixel is between 0 and 1.

Για την τροφοδότηση των εικόνων στο μοντέλο είναι απαραίτητη μια διαδικασία επεξεργασίας των δεδομένων. Η διαδικασία αποτελείτε από τα εξής βήματα:

1. Αλλαγή των διαστάσεων των εικόνων διότι το μοντέλο απαιτεί κάθε εικόνα να περιέχει τον αριθμό των καναλιών. Παρότι οι εικόνες του σετ δεδομένων είναι επιπέδου γκρίζου δηλαδή έχουν μόνο ένα κανάλι θα πρέπει να προσθέσουμε τον αριθμό των καναλιών στις διαστάσεις της εικόνας.
2. Το μοντέλο δέχεται εικόνες μεγέθους 32x32 εικονοστοιχείων για αυτό τον λόγο θα προσθέσουμε μηδενικά εικονοστοιχεία έτσι ώστε το μέγεθος των εικόνων να γίνει 32x32.
3. Επίσης η κανονικοποίηση των δεδομένων βοηθάει στην απόδοση του μοντέλου καθώς η διαδικασία της εκπαίδευσης του μοντέλου συμπεριλαμβάνει τον πολλαπλασιασμό των τιμών με τα βάρη (weights) και την πρόσθεση της κλίσης (bias) για να εκτελεστεί η ενεργοποίηση και να οπισθοτροφοδοτηθεί στο δίκτυο το διαφορικό για να εκπαιδευτεί το μοντέλο. Για αυτό τον λόγο έχοντας όλες τις τιμές στο ίδιο εύρος αυξάνεται η αποδοτικότητα του δικτύου.
4. Κωδικοποίηση των κατηγοριών με την μέθοδο One Hot Encoding. Η μέθοδος One Hot Encoding χρησιμοποιείτε για να μετατρέψει κατηγορικά δεδομένα σε μια μορφή που θα μπορεί να τροφοδοτηθεί σε ένα νευρωνικό δίκτυο. Η κατηγορικές τιμές αντιπροσωπεύουν μια κατηγορία με έναν αριθμό. Για παράδειγμα στο σετ δεδομένων Fashion MNIST ο κατηγορικός αριθμός 3 αντιστοιχεί στην κατηγορίαdress. Το πρόβλημα με αυτή την προσέγγιση είναι ότι το μοντέλο θεωρεί πως όσο μεγαλύτερος ο αριθμός που αντιστοιχεί σε μια κατηγορία τόσο καλύτερη είναι η κατηγορία το οποίο δεν ισχύει και οδηγεί σε λάθος εκπαίδευση του μοντέλου. Για αυτό τον λόγο χρησιμοποιείται η κωδικοποίηση one hot encoding η οποία εκτελεί την διαδικοποίηση των κατηγοριών για κάθε εικόνα αντιστοιχεί την τιμή 1 στην κατηγορία που ανήκει η εικόνα και την τιμή 0 στις κατηγορίες που δεν ανήκει η εικόνα. Κατά αυτόν τον τρόπο αυξάνεται το μέγεθος των παραμέτρων των εικόνων. Έχοντας μόνο μια παράμετρο που κρατάει την κατηγορία που ανήκει η εικόνα ύστερα από αυτή την διαδικασία η εικόνα θα έχει 10 παραμέτρους όσες και οι κατηγορίες του σετ δεδομένων.



2.2.2 Μοντέλο LeNet-5

Έχοντας εκτελέσει την απαραίτητη διαδικασία της προεπεξεργασίας των δεδομένων η συνέχεια είναι να οριστεί η αρχιτεκτονική του δικτύου. Το δίκτυο το οποίο εκπαιδεύτηκε στα πλαίσια της διπλωματικής εργασίας είναι το LeNet-5.

Το δίκτυο LeNet-5 είναι ένα απλό δίκτυο σε σχέση με τα πιο σύγχρονο δίκτυα τα οποία είναι πιο περίπλοκα και αποτελούνται από περισσότερα επίπεδα. Το δίκτυο αποτελείτε από 7 επίπεδα, τα 3 από τα οποία είναι συνελικτικά επίπεδα (convolution layer), τα δυο είναι επίπεδα εξαγωγής (pooling layer)ένα πλήρως συνδεδεμένο επίπεδο (fully connected layer) το οποίο ακολουθείτε από ένα επίπεδο εξόδου (output layer).

Η είσοδος του πρώτου επίπεδο είναι μια εικόνα μεγέθους 32x32 εικονοστοιχείων. Το επίπεδο έχει 6 φίλτρα μεγέθους 5x5 τα οποια περνάνε από την εικόνα με βήμα 1 και επέκταση (padding) 0. Εύκολα μπορεί να υπολογιστεί από τον τύπο του ύψους και του πλάτους της συνέλιξης το μέγεθος της εξόδου του επιπέδου.

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

βάζοντας τις τιμές των παραμέτρων στις παραπάνω εξισώσεις βρίσκομαι πως το ύψος της εικόνας της εξόδου θα είναι (32 - 5 + 2 * 0) / 1 + 1 δηλαδή 32 - 5 + 1 άρα 28. Παρομοίως υπολογίζεται το πλάτος της εικόνας της εξόδου του επιπέδου (32 - 5 + 2 * 0) / 1+ 1 άρα 28. Άρα η έξοδος του πρώτου συνελικτικου επιπέδου θα είναι (28,28, 6).

Το δεύτερο επίπεδο είναι ένα επίπεδο εξαγωγής. Περνώντας δηλαδή μια μάσκα εξαγωγής μέσης τιμής μεγέθους 2x2 με βήμα 2 από τις εικόνες της εξοδου του πρώτου επιπέδου θα μικρύνει το μέγεθος των εικόνων ο αριθμός των καναλιών όμως θα παραμήνει ίδιος. Το μέγεθος της εξόδου του επιπέδου εξαγωγής υπολογίζεται από τους εξής τύπους:

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

βάζοντας τις τιμές των παραμέτρων στις παραπάνω εξισώσεις βρίσκομαι πως το ύψος της εικόνας της εξόδου θα είναι (28 - 2 + 2 * 0) / 2 + 1 δηλαδή 13 + 1 άρα 14. Παρομοίως υπολογίζεται το πλάτος της εικόνας της εξόδου του επιπέδου (28 - 2 + 2 * 0) / 2 + 1 δηλαδή 13 + 1 άρα 14. Άρα η έξοδος του δεύτερου επιπέδου εξαγωγής θα είναι (14, 14, 6).

Το τρίτο επίπεδο του δικτύου LeNet-5 είναι ακόμη ένα επίπεδο συνέλιξης. Το συγκεκριμένο επίπεδο αποτελείτε από 16 φίλτρα μεγέθους 5x5 τα οποια περνάνε από τις εικόνες με βήμα 1. Στο συγκεκριμένο μόνο 10 από τα 16 φίλτρα είναι συνδεδεμένα στην έξοδο του προηγούμενου επιπέδου όπως φαίνεται στην παρακάτω εικόνα.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED

Ο κύριος λόγος είναι για να σπάσει η συμμετρία στο δίκτυο και να κρατηθεί ο αριθμός των συνδέσεων σε λογικά πλαίσια. Το μέγεθος της εξόδου πάλι θα υπολογιστεί από τον τύπο του υπολογισμού του πλάτους και τους ύψους. Το ύψος θα είναι (14 - 5) / 1 + 1 δηλαδή 9 + 1 άρα 10 και το πλάτος θα είναι πάλι (14 - 5) / 1 + 1 δηλαδή 9 + 1 άρα 10. Άρα το μέγεθος της εξόδου του τρίτου συνελικτικού επιπέδου θα είναι (10, 10 , 16).

Το τέταρτο επίπεδο είναι πάλι ένα επίπεδο εξαγωγής μέσης τιμής που αποτειλειτε απο φίλτρο μεγέθους 2x2 τα οποια περνάνε από τις εικόνες με βήμα 2. Το μέγεθος της εξόδου του συγκεκριμένου επίπεδο μπορείς να υπολογιστεί εύκολα από τους τύπους υπολογισμού πλάτους και ύψους. Το ύψος θα είναι (10 - 2) / 2 + 1 δηλαδή 4 + 1 άρα 5 και το πλάτος θα είναι πάλι (10 - 2) / 2 + 1 δηλαδή 5 + 1 άρα 5. Άρα το μέγεθος της εξόδου του τρίτου συνελικτικού επιπέδου θα είναι (5, 5 , 16).
Το πέμπτο επίπεδο είναι ένα πλήρως συνδεδεμένο συνελικτικο επίπεδο που αποτελείτε από 120 φίλτρα μεγέθους 5x5. Κάθε ένα μια από τις 120 σημεία είναι πλήρως συνδεδεμένα με τα 400 σημεία του προηγουμενου επιπέδου. Το μέγεθος της εξόδου του συγκεκριμένου επίπεδο μπορείς να υπολογιστεί εύκολα από τους τύπους υπολογισμού πλάτους και ύψους. Το ύψος θα είναι (5 - 5) / 1 + 1 δηλαδή 0 + 1 άρα 1 και το πλάτος θα είναι πάλι (5 - 5) / 1 + 1 δηλαδή 0 + 1 άρα 1. Άρα το μέγεθος της εξόδου του τρίτου συνελικτικού επιπέδου θα είναι (1, 1 , 120).

Το έκτο επίπεδο είναι ένα πλήρως συνδεδεμένο επίπεδο που αποτελείτε από 84 σημεία. Και το τελευταίο επίπεδο είναι ένα πλήρως συνδεδεμένο softmax επίπεδο που βγάζει στην έξοδο 10 που κάθε μια αντιστοιχεί στην πιθανότητα της εικόνας να ανοίκει σε μία από τις 10 κλάσεις του σετ δεδομένων.

Έχοντας μελετήσει το μοντέλο LeNet 5 συνεχίζουμε το πρόγραμμα δηλώνοντας το δίκτυο.

In [8]:

```
# Define the model architecture
layer1={}
layer1['mode'] = 'conv'
layer1['filters'] = 5
layer1['n_C_prev'] = 1
layer1['n_C'] = 6
layer1['pad'] = 0
layer1['stride'] = 1
layer2={}
layer2['mode'] = 'pool'
layer2['filters'] = 2
layer2['stride'] = 2
layer3={}
layer3['mode'] = 'conv'
layer3['filters'] = 5
layer3['n_C_prev'] = 6
```

Όπως φαίνεται στον παραπάνω κώδικα, κάθε επίπεδο είναι ένα Python dictionary στο οποίο αποθηκεύονται οι παράμετροι του κάθε επιπέδου. Στην συνέχεια δημιουργείτε μια λίστα από όλα τα επίπεδα.

2.2.3 Εκπαίδευση Μοντέλου

Το σημαντικότερο βήμα είναι η εκπαίδευση του μοντέλου. Έχοντας εισάγει την βιβλιοθήκη cnn_utils που δημιουργήθηκε στα πλαίσια της πτυχιακής εργασίας. Η βιβλιοθήκη περιέχει όλες τις απαραίτητες μεθόδους για την εκπαίδευση του μοντέλου. Καλώντας την μέθοδο train και δίνοντας τις απαραίτητες παραμέτρους πραγματοποιείται η εκπαίδευση του μοντέλου.

In [9]:
```
initial_layers_path = 'data/initial_layers_{}.npy'.format(index+1)
initial_layers = functions.initialize_layers(construct_layers)
np.save(initial_layers_path, initial_layers)
print('----------------------------------------------------------------------------------------------------')

layers, train_acc, test_acc = functions.train(X_train, Y_train, X_test, Y_test, initial_layers,
                                              num_exp=index, batch_size=100, num_epoch=10, learning_rate=0.1)
print('\n')

```

Με τον παραπάνω τρόπο καλείτε η μέθοδος train. Και δίνει ως έξοδο το εκπαιδευμένο μοντέλο, μια λίστα με την ακρίβεια εκπαίδευσης του μοντέλου για κάθε epoch καθώς και μια λίστα με την ακρίβεια επαλήθευσης για κάθε epoch.
Επίσης εκτυπώνετε η δομή του δικτύου.

```
_____________________________________
Layer (type)          Output Shape
=====================================
conv                (100, 28, 28, 6)
_____________________________________
pool                (100, 14, 14, 6)
_____________________________________
conv                (100, 10, 10, 16)
_____________________________________
pool                (100, 5, 5, 16)
_____________________________________
conv                (100, 1, 1, 120)
_____________________________________
fc                  (84, 100)
```

Επίσης τυπώνετε ο χρόνος για κάθε epoch καθώς και ο συνολικός χρόνος.

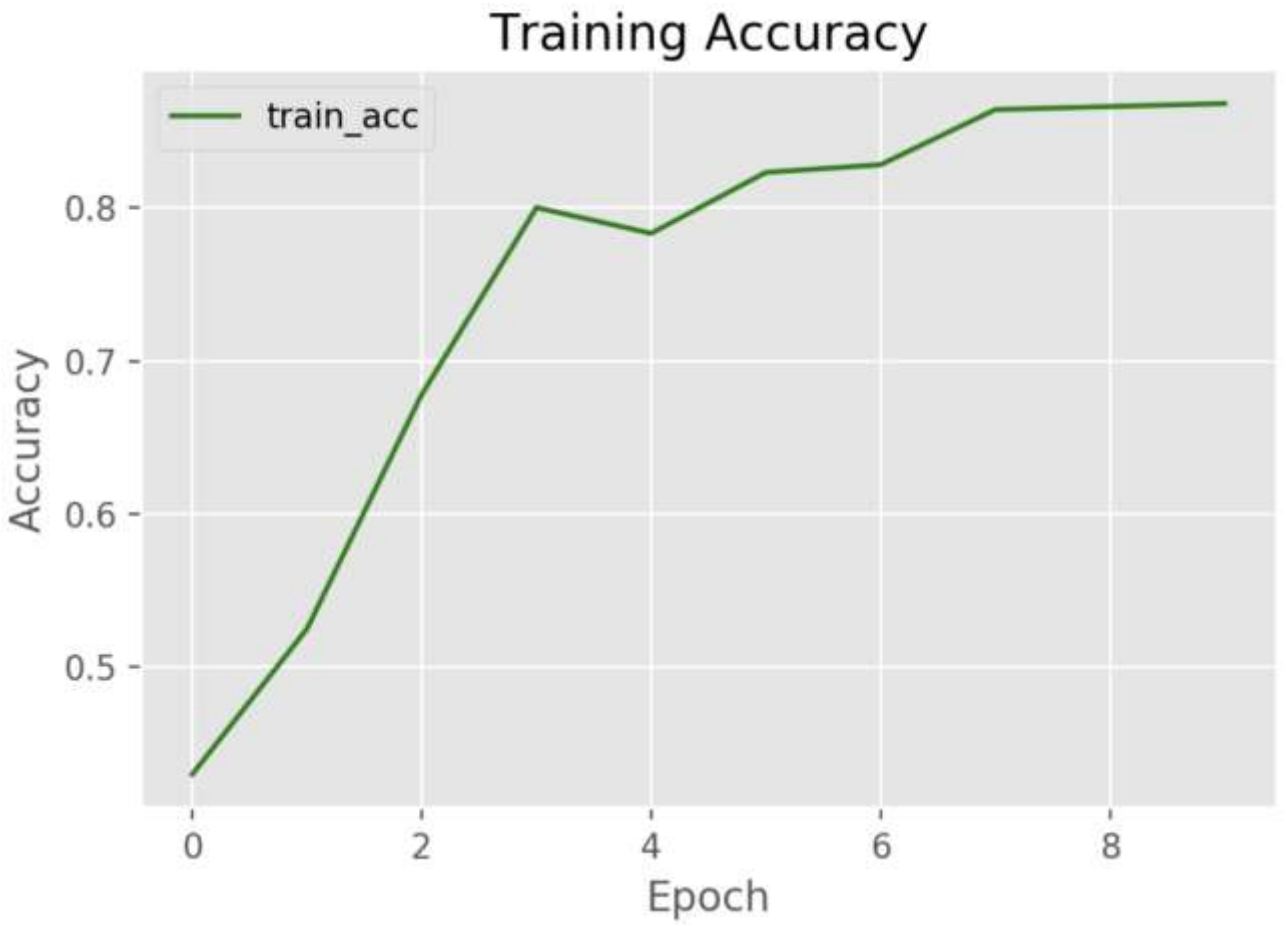
```
Epoch : 1:  100%|██████████| 600/600 [03:46:06<00:00, 22.66s/it]
Epoch : 2:  100%|██████████| 600/600 [04:46:12<00:00, 28.60s/it]
Epoch : 3:  100%|██████████| 600/600 [04:33:41<00:00, 27.30s/it]
Epoch : 4:  100%|██████████| 600/600 [04:51:52<00:00, 29.17s/it]
Epoch : 5:  100%|██████████| 600/600 [03:44:14<00:00, 22.45s/it]
Epoch : 6:  100%|██████████| 600/600 [04:01:22<00:00, 24.12s/it]
Epoch : 7:  100%|██████████| 600/600 [03:52:53<00:00, 23.24s/it]
Epoch : 8:  100%|██████████| 600/600 [04:25:58<00:00, 26.51s/it]
Epoch : 9:  100%|██████████| 600/600 [03:34:07<00:00, 21.40s/it]
Epoch : 10: 100%|██████████| 600/600 [04:03:42<00:00, 24.37s/it]

Total time elapsed: 149856 seconds
```

Με βάση τα παραπάνω αποτελέσματα ο μέσος όρος της διάρκειας της κάθε επανάληψης προκύπτει 24.976 δευτερόλεπτα . Ο μέσος όρος διάρκειας των epochs είναι 14985.6 δευτερόλεπτα δηλαδή 4 ώρες και 10 λεπτά.Η συνολική διάρκεια της πλήρης εκπαίδευσης του μοντέλου στις 60000 εικόνες με batch size 100 και 10 epochs είναι 149845 δευτερόλεπτα δηλαδή 41 ώρες και 48 λεπτά.

2.2.4 Μελέτη αποτελεσμάτων του μοντέλου

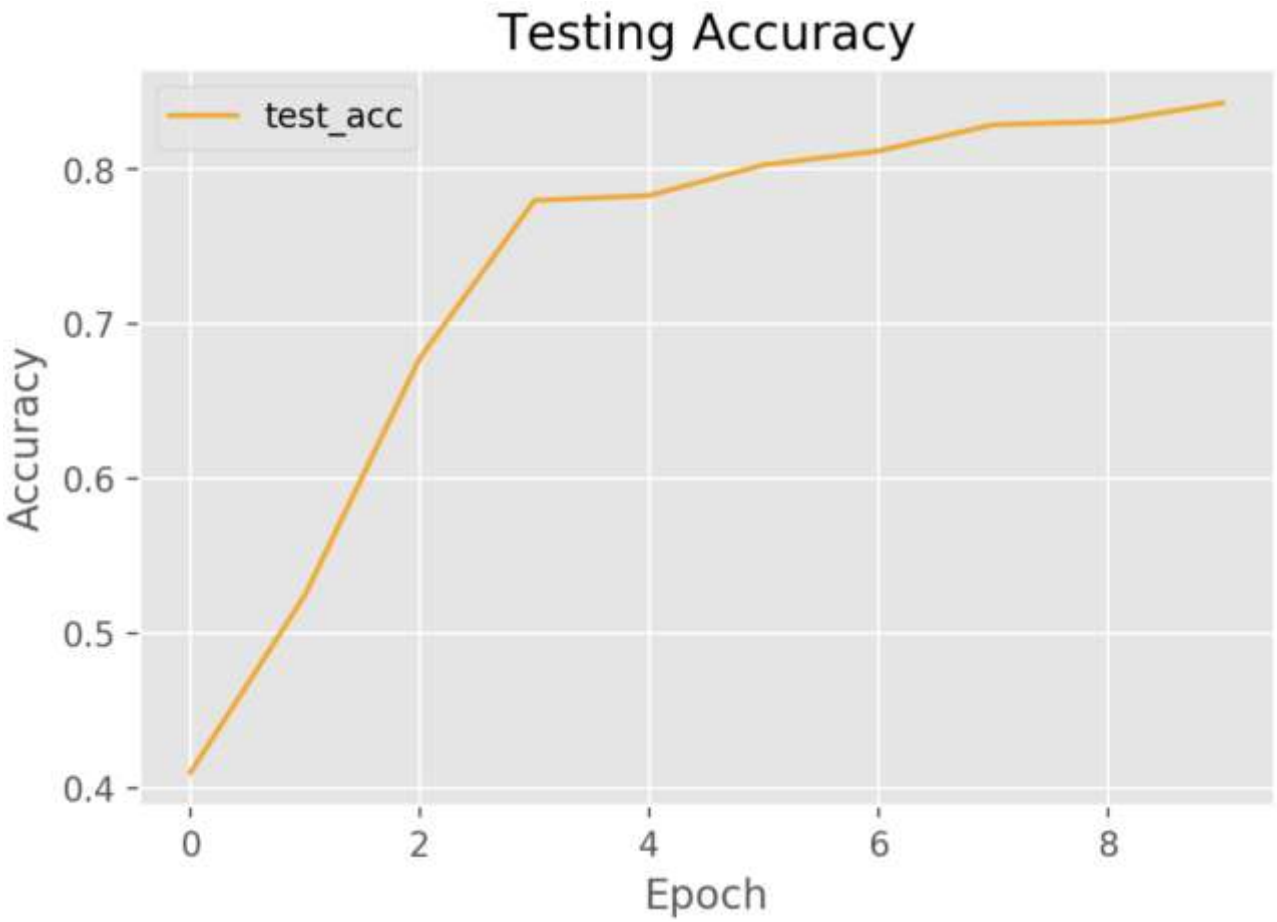
Χρησιμοποιώντας την βιβλιοθήκη matplotlib θα δημιουργηθούν διαγράμματα για να μελετηθεί η μεταβολή της ακρίβειας στα epochs. Αρχικά θα δημιουργηθεί το διάγραμμα ακρίβειας για το σετ δεδομένων εκπαίδευσης.

In [65]:
```
num_epochs = np.arange(0, 10)
plt.figure(dpi=200)
plt.style.use('ggplot')
plt.plot(num_epochs, train_acc, label='train_acc', c='green')
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('plot.png')
```



Στο παραπάνω διάγραμμα φαίνεται η μεταβολή την ακριβείας για το σετ δεδομένων εκπαίδευσης. Στο πρώτο epoch η ακρίβεια ξεκινάει από χαμηλές τιμές ελάχιστα μεγαλύτερες του 0.4. Ύστερα από δέκα epochs η τιμή της ακριβείας του σετ δεδομένων εκπαίδευσης αυξάνεται αρκετά πλησιάζοντας αρκετά τα εννέα δέκατα της μονάδας.

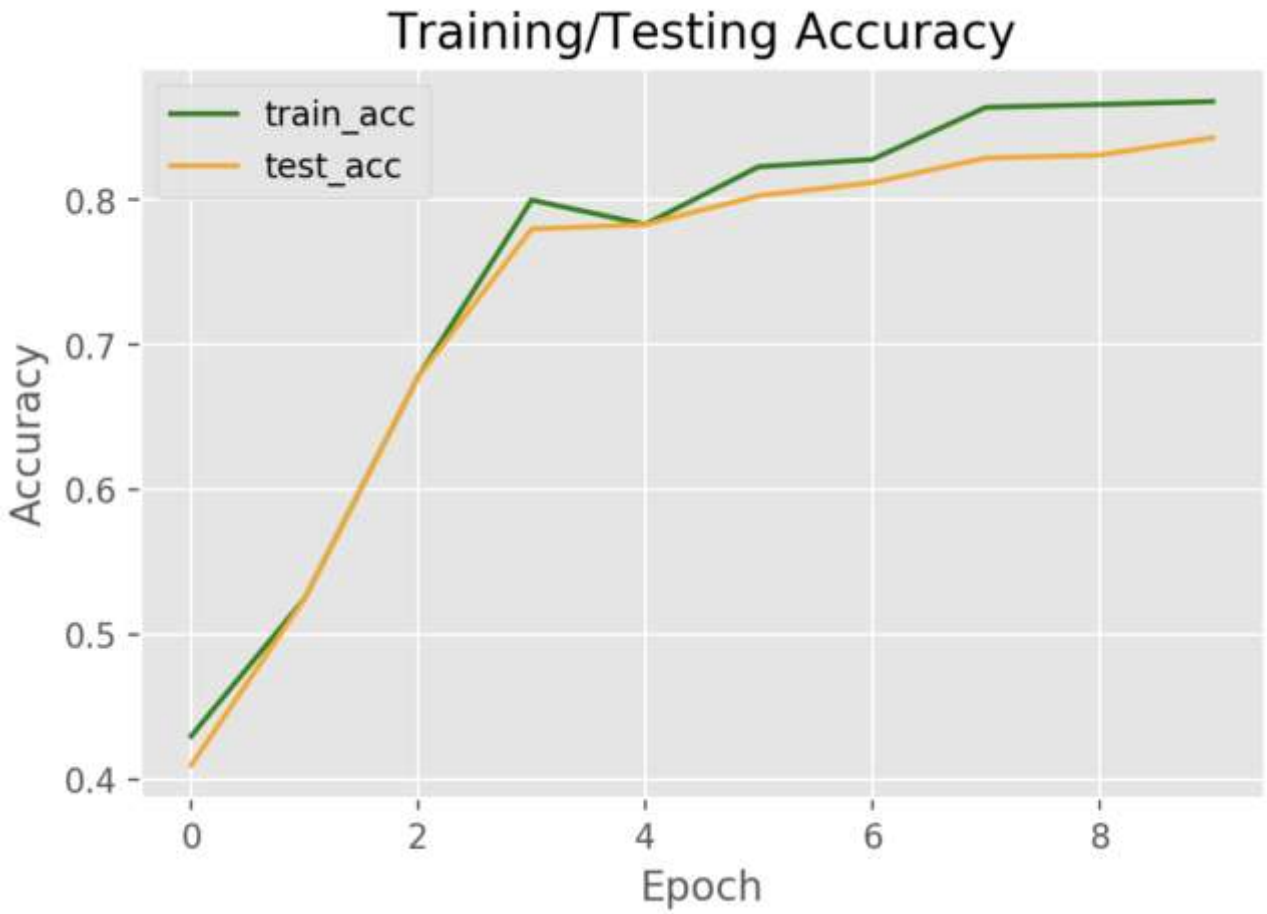
Στην συνέχεια θα μελετηθεί η ακρίβεια στο σετ δεδομένων επαλήθευσης. Τα αποτελέσματα αναμένονται να είναι ελάχιστα χαμηλότερα από την ακρίβεια του σετ δεδομένων εκπαίδευσης. Με παρόμοιο τρόπο χρησιμοποιώντας την βιβλιοθήκη matplotlib δημιουργείτε το διάγραμμα ακριβείας για το σετ δεδομένων επαλήθευσης.



Όπως φαίνεται στο παραπάνω διάγραμμα η τιμή στο πρώτο epoch ξεκάνει από πολύ χαμηλά προσεγγίζοντας σχεδόν το 0.4. Έπειτα από την εκπαίδευση σε 10 epochs η τιμή της ακρίβειας του σετ δεδομένων επαλήθευσης αυξάνεται και ξεπερνάει για πολύ το 0,8.

Επίσης συγκριθεί η ακρίβεια του σετ δεδομένων εκπαίδευσης με το σετ δεδομένων επαλήθευσης.

Accuracy Comparison	
Train	Test
0.430	0.410
0.525	0.525
0.678	0.678
0.800	0.780
0.783	0.783
0.823	0.803
0.828	0.812
0.864	0.829
0.866	0.831
0.868	0.843



Όπως φαίνεται στο παραπάνω διάγραμμα καθώς και στον πίνακα με τις τιμές της ακρίβειας η τιμές στο σετ δεδομένων εκπαίδευσης είναι ελάχιστα μεγαλύτερες από τις τιμές στο σετ δεδομένων επαλήθευσης. Το φαινόμενο αυτό οφείλετε στο γεγονός ότι το στο μοντέλο τροφοδοτείτε το σετ δεδομένων εκπαίδευσης και για αυτόν τον λόγο το μοντέλο προσαρμόζεται στο συγκεκριμένο σετ παρουσιάζοντας έτσι ελάχιστα καλύτερα αποτελέσματα από το σετ επαλήθευσης το οποίο αποτελείτε από εικόνες που δεν έχουν τροφοδοτηθεί στο μοντέλο στο παρελθόν.

2.2.5 Σύγκριση με μοντέλο Keras

Για να συγκριθεί η προσέγγιση της συγκεκριμένης πτυχιακής εργασίας με πιο εδραιωμένες προσεγγίσεις που χρησιμοποιούνε ευρύτατα στον τομέα θα υλοποιηθεί το ίδιο μοντέλο LeNet-5 με την βιβλιοθήκη Keras.

Αρχικά θα εισαχθούν τα απαρέτητα τμήματα της βιβλιοθήκης Keras.

```
In [110]:
1 import keras
2 import keras.layers as layers
3 from keras.models import Sequential
4 from keras.preprocessing.image import ImageDataGenerator
5 from keras.utils.np_utils import to_categorical
```

Στην συνέχεια θα οριστεί το μοντέλο LeNet-5. Όπως και με την βιβλιοθήκη που δημιουργήθηκε στα πλαίσια της πτυχιακής εργασίας θα πρέπει να ορισθούν ένα ένα τα επίπεδα με τις παραμέτρους τους.

```
In [116]:
1  model = keras.Sequential()
2
3  model.add(layers.Conv2D(filters=6, kernel_size=5, strides= 1, activation='relu',
4                          input_shape=(32,32,1)))
5  model.add(layers.AveragePooling2D())
6
7  model.add(layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
8  model.add(layers.AveragePooling2D())
9
10 model.add(layers.Flatten())
11
12 model.add(layers.Dense(units=120, activation='relu'))
13
14 model.add(layers.Dense(units=84, activation='relu'))
15
16 model.add(layers.Dense(units=10, activation = 'softmax'))
```

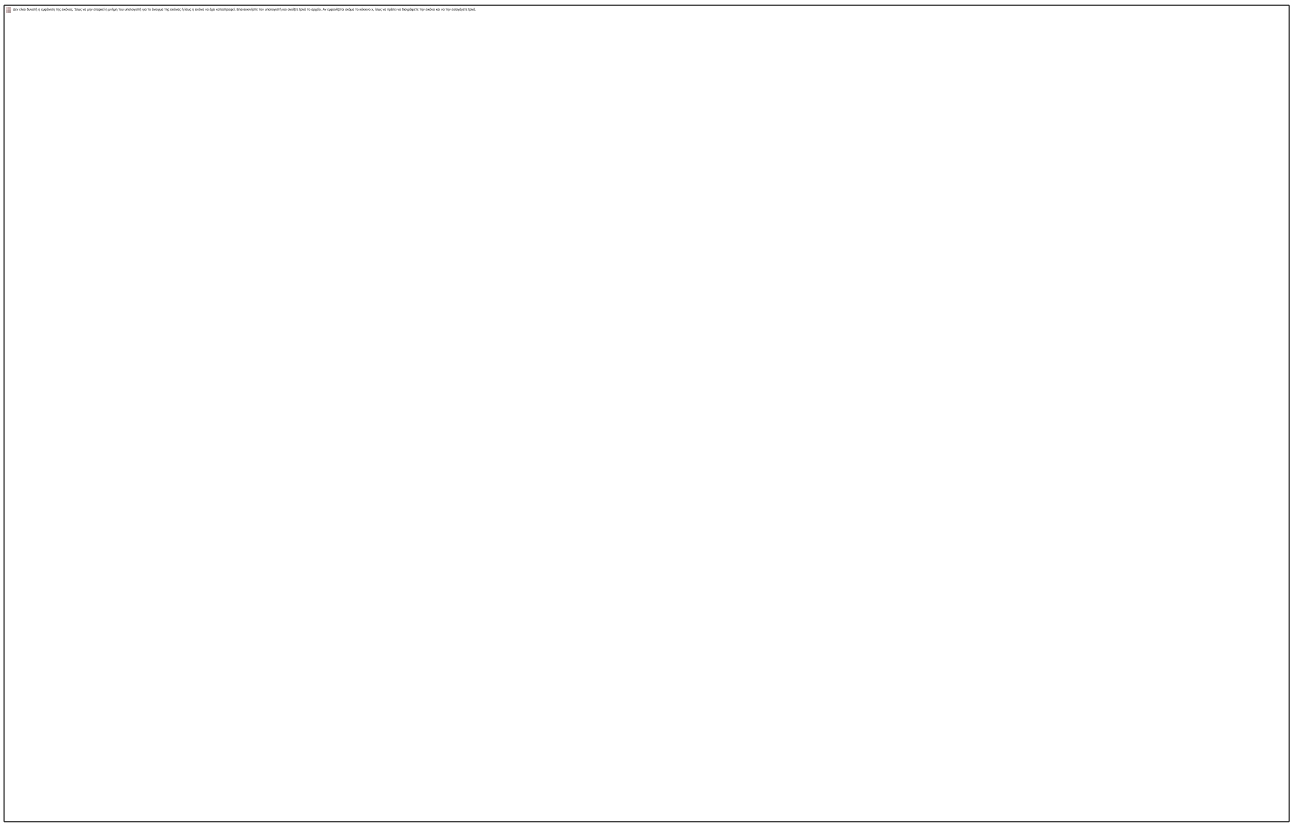
Η μέθοδος summary() του Keras δίνει την δυνατότητα της εμφανίσεις της δομής του δικτύου.

```
In [117]:
1 model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_5 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_6 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_7 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_8 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_9 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

Όπως φαίνεται στον παραπάνω πίνακα ορίσαμε το μοντέλο χρησιμοποιώντας 8 επίπεδα. Το πρώτο, το τρίτο επίπεδο είναι επίπεδα συνέλιξης το δεύτερο και το τέταρτο είναι επίπεδα εξαγωγής μέσης τιμής το πέμπτο επίπεδο είναι επίπεδο εξομάλυνσης της λίστας και τα τελευταία 3 επίπεδα είναι πλήρως συνδεδεμένα επίπεδα.

Στη συνέχεια θα δηλωθούν το μέγεθος του batch καθώς και ο αριθμός των epochs. Θα χρησιμοποιηθούν οι ιδές τιμές με την προηγούμενη προσέγγιση για να είναι ξεκάθαρη η σύγκριση.



Όπως φαίνεται στην παραπάνω εικόνα ο μέσος χρόνος για κάθε epoch είναι 26 δευτερόλεπτα και ο συνολικός χρόνος είναι λιγότερος από 5 λεπτά είναι δραματικά μικρότερος σε σχέση με την χρήση της βιβλιοθήκης cnn_utils που ήταν 41 ώρες και 48 λεπτά. Η διαφορά αυτή οφείλετε στο γεγονός ότι η βιβλιοθήκη Keras είναι γραμμένη σε γλώσσα χαμηλότερου επιπέδου και χρησιμοποιεί διάφορες τεχνικές για να αύξηση την αποτελεσματικότητα και την ταχύτητα της εκπαίδευσης του μοντέλου.Επίσης η βιβλιοθήκη Keras χρησιμοποιείτε αρκετά χρόνια και έχουν δουλέψει αρκετοί ερευνητές στην αύξηση της αποτελεσματικότητας της.

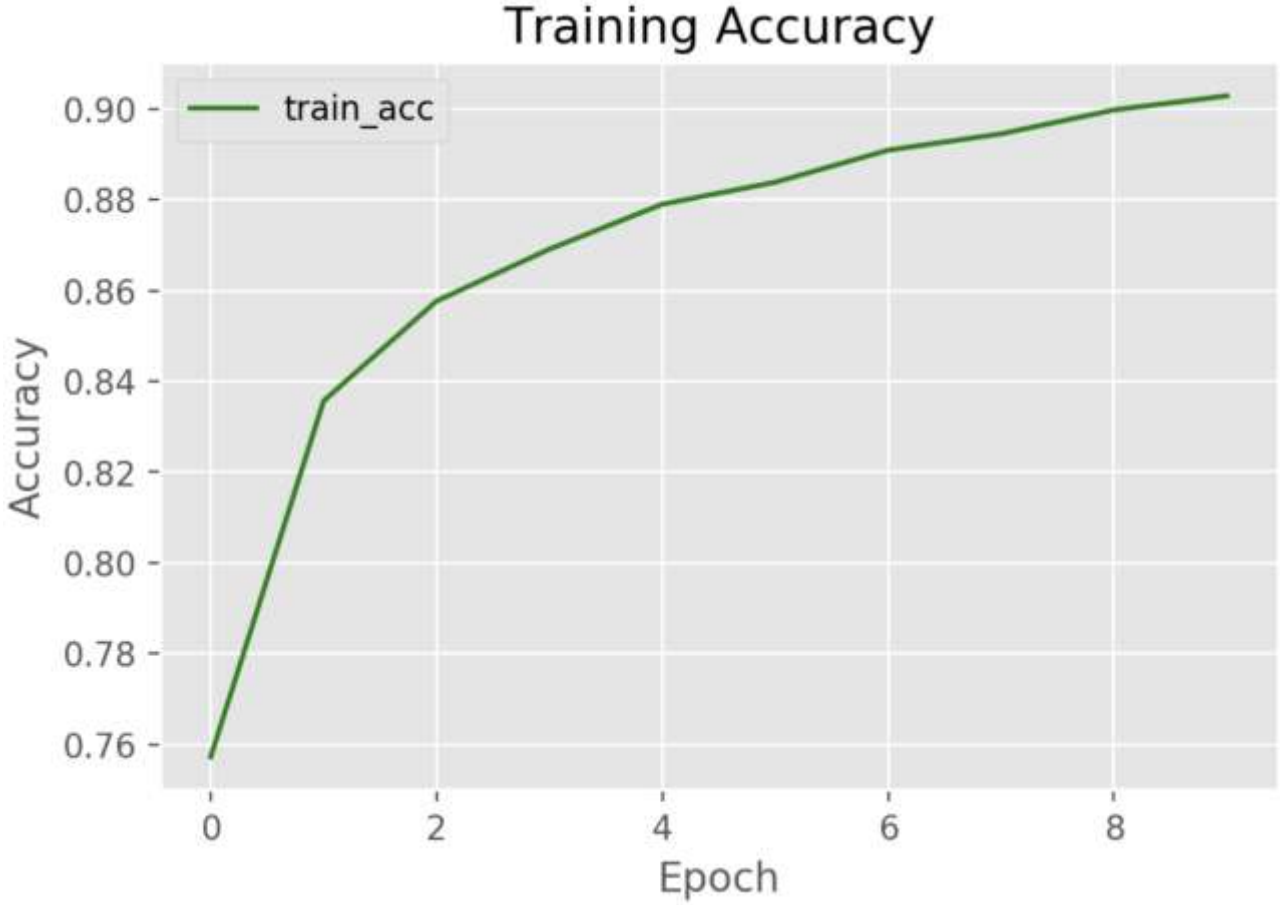
Το τελικό αποτέλεσμα της ακριβείας για το σετ δεδομένων επαλήθευσης χρησημοποιοντας την βιβλιοθήκη Keras είναι 0.89 όπως φαίνεται και στην παρακάτω εικόνα.

```
In [11]:
1 score = model.evaluate(X_test, Y_test)
2 print('Test loss:', score[0])
3 print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 1s 71us/step
Test loss: 0.30872519487142563
Test accuracy: 0.8907999992370605
```

Η τιμή της ακρίβειας για το σετ δεδομένων επαλήθευσης είναι αρκετά κοντά στην τιμή της βιβλιοθήκης cnn_utils που ήταν 0.843 παρόλο που για την εκπαίδευση με την βιβλιοθήκη Keras χρειαστήκαν 41 ώρες και 43 λεπτά λιγότερα.

Με παρόμοιο τρόπο όπως και στην βιβλιοθήκη cnn_utils θα δημιουργηθεί ένα διάγραμμα που θα δείχνει την μεταβολή των τιμών της ακρίβειας κατά την εκπαίδευση.



2.2.6 Σύγκριση με άλλο μοντέλο

Στα πλαίσια της πτυχιακής εργασίας θα δημιουργηθεί ακόμα ένα μοντέλο για να συγκριθεί με το προηγούμενο. Το μοντέλο θα είναι πιο μεγάλο από το LeNet-5 και θα αποτελείτε από 8 επίπεδα σε σχέση με τα 7 του LeNet-5. Το μοντέλο είναι μια παραλλαγή του VGGNet χρησιμοποιώντας λιγότερα επίπεδα. Το μοντέλο θα χρησιμοποιεί συνέλιξης μεγέθους 3x3. Το μοντέλο θα αποτελείτε 4 συνελικτικά επίπεδα που χρησιμοποιούν την ενεργοποίηση RELU, 2 επίπεδα εξαγωγής μεγίστου καθώς και ένα επίπεδο ένα πλήρως συνδεδεμένο επίπεδο καθώς και το επίπεδο εξόδου softmax. Παρακάτω θα οριστεί η αρχιτεκτονική του δικτύου.

In [7]:

```
model.add(Conv2D(32, (3, 3), padding="same",
    input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(classes))
model.add(Activation("softmax"))
```

Η δομή του δικτύου φαίνεται αναλυτικά στον παρακάτω πίνακα.

```
In [8]: 1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2097664
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 2,170,602
Trainable params: 2,169,194
Non-trainable params: 1,408
```

Στη συνέχεια θα εκπαιδεύσουμε το μοντέλο με το ίδιο dataset και με τις ιδές παραμέτρους δηλαδή μέγεθος batch 100 και 10 epochs.

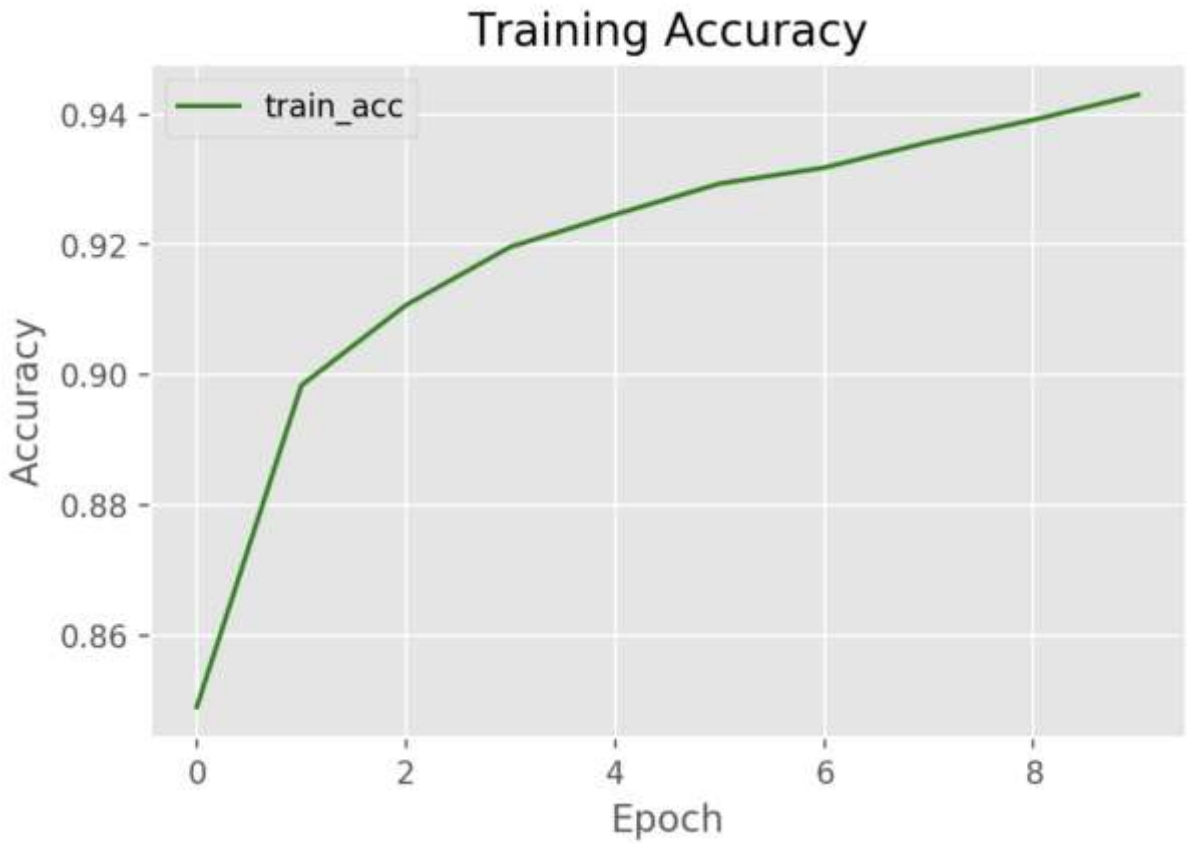
```
Epoch 1/10
600/600 [==============================] - 232s 387ms/step - loss: 0.4408 - accuracy: 0.8490
Epoch 2/10
600/600 [==============================] - 241s 401ms/step - loss: 0.2835 - accuracy: 0.8984
Epoch 3/10
600/600 [==============================] - 235s 392ms/step - loss: 0.2473 - accuracy: 0.9107
Epoch 4/10
600/600 [==============================] - 229s 381ms/step - loss: 0.2211 - accuracy: 0.9197
Epoch 5/10
600/600 [==============================] - 229s 382ms/step - loss: 0.2075 - accuracy: 0.9247
Epoch 6/10
600/600 [==============================] - 231s 386ms/step - loss: 0.1928 - accuracy: 0.9294
Epoch 7/10
600/600 [==============================] - 235s 392ms/step - loss: 0.1839 - accuracy: 0.9319
Epoch 8/10
600/600 [==============================] - 230s 384ms/step - loss: 0.1748 - accuracy: 0.9358
Epoch 9/10
600/600 [==============================] - 216s 360ms/step - loss: 0.1644 - accuracy: 0.9392
Epoch 10/10
600/600 [==============================] - 221s 368ms/step - loss: 0.1534 - accuracy: 0.9431
```

Όπως φαίνεται στην παραπάνω φωτό ο μέσος χρόνος για ένα epoch είναι 235 δευτερόλεπτα δηλαδή 3 λεπτά και 55 δευτερόλεπτα που είναι αρκετά περισσότερο από το δίκτυο LeNet-5 εκπαιδευμένο με την βιβλιοθήκη Keras. Το αποτέλεσμα είναι αναμενόμενο καθώς το δίκτυο χρησιμοποιεί μεγαλύτερο αριθμό επιπέδων.

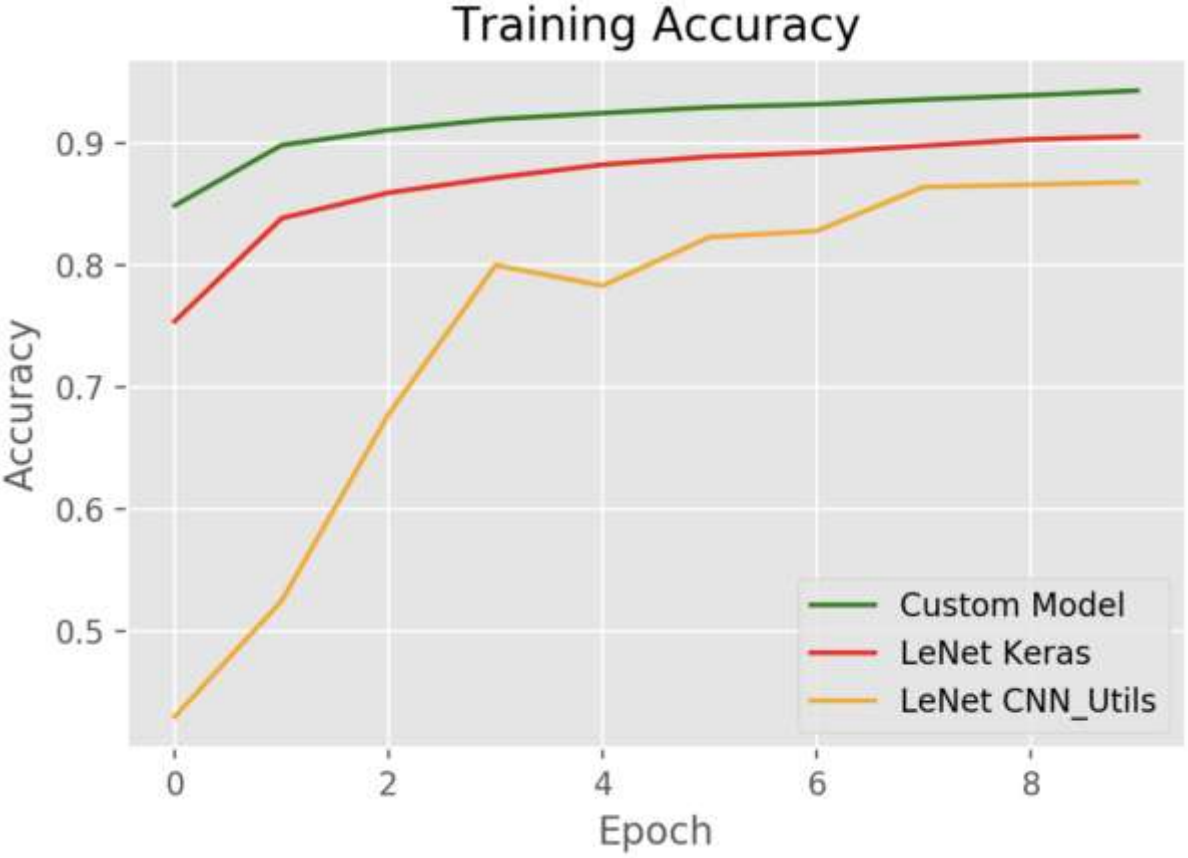
```
In [15]:
1 score = model.evaluate(X_test, Y_test)
2 print('Test loss:', score[0])
3 print('Test accuracy:', score[1])

10000/10000 [==============================] - 8s 827us/step
Test loss: 0.20018988428413867
Test accuracy: 0.9280999898910522
```

Το τελικό αποτέλεσμα της ακρίβειας του μοντέλου στο σετ δεδομένων επαλήθευσης είναι 0.928 όπως φαίνεται στην παραπάνω φωτό. Η ακρίβεια του συγκεκριμένου μοντέλου είναι μεγαλύτερη από την ακρίβεια του δικτύου LeNet-5 που ήταν 0.89.

Επίσης και για αυτό το μοντέλο θα δημιουργηθεί διάγραμμα για να μελετηθεί η μεταβολή της ακρίβειας κατά την εκπαίδευση.



Στη συνέχεια θα δημιουργηθεί ένα διάγραμμα που θα περιλαμβάνει τις τιμές για την ακρίβεια και των τριών μοντέλων για το σετ δεδομένων εκπαίδευσης.



Όπως φαίνεται στο παραπάνω διάγραμμα το τελευταίο μοντέλο που δημιουργήθηκε στα πλαίσια της πτυχιακής είναι πιο αποδοτικό από το LeNet. Επίσης φαίνεται ότι το LeNet της βιβλιοθήκης Keras είναι πιο αποδοτικό από το μοντέλο LeNet της βιβλιοθήκης CNN_Utils που δημιουργήθηκε στα πλαίσια της πτυχιακής εργασίας.

2.3 Υλοποίηση Βιβλιοθήκης CNN_Utils

Στα πλαίσια της πτυχιακής εργασίας υλοποιήθηκε μια βιβλιοθήκη για την δημιουργία και εκπαίδευση συνελικτικών νευρωνικών δικτύων. Η βιβλιοθήκη αποτελείτε από 23 συναρτήσεις και υλοποιεί κάθε λειτουργιά ενός συνελικτικου νευρωνικού δικτύου. Περνώντας ως παράμετρο την αρχιτεκτονική του νευρωνικού δίκτυου καθώς και το σετ δεδομένων εκπαίδευσης δημιουργείτε ένα συνελικτικό νευρωνικό δίκτυο

2.3.1 Αρχικοποίηση τιμών επιπέδων

```
def sigmoid_activation(Z):
    '''
    Calculates the value of sigmoid functions

    Arguments:
    Z -- A numeric value

    Returns:
    A -- The result of the sigmoid function calculation for the Z
    cache -- The initial value of Z
    '''
    #Sigmoid function
    A = 1/(1+np.exp(-Z))
    cache = Z
    return A, cache
```

Ο σκοπός της συνάρτησης initialize_layers(layers) είναι να αρχικοποιήσει τις τιμές των επιπέδων. Παίρνει σαν είσοδο μια λίστα από αντικείμενα που αντιστοιχούν στα επίπεδα του μοντέλου και στις αντίστοιχες παραμέτρους. Οι πιθανοί τύποι των επιπέδων είναι οι εξής: pooling layer, fully connected layer και convolution layer. Στην περίπτωση του pooling layer δεν χρειάζεται να γίνει αρχικοποίηση τιμών. Στην περίπτωση του fully connected layer (Πλήρως συνδεδεμένο επίπεδο) αρχικοποιεί τα βάρη (weight) καθώς και οι κλίσεις (bias) . Επίσης αριχκοποιούμε τα dW και db,που είναι οι διαφορές των βαρών και των κλίσεων με το προηγούμενο επίπεδο, με πίνακες από μηδενικά που έχουν το ίδιο μέγεθος με τους πίνακες από τα βάρη και τις κλίσεις. Τέλος η συνάρτηση επιστρέφει την καινούργια λίστα με τις αρχικοποιημέμες τιμές των επιπέδων.

2.3.2 Σιγμοειδής συνάρτηση ενεργοποίησης

Ο σκοπός της συνάρτησης sigmoid_activation(Z) είναι να υπολογίσει και να επιστρέψει την

```
def initialize_layers(layers):
    """
    Initialize the layer parameters

    Arguments:
    layers -- A list of element where each element defines the type of layer and the hyperparameters.

    Returns:
    new_layers -- The list of layers with the initialized values.
    """
    new_layers = []
    for i, layer in enumerate(layers):
        mode = layer['mode'] # 'fc', 'conv', 'pool'
        # Pooling layer
        if mode == 'pool':
            new_layers.append(layer)
            continue
        # Fully Connected layer
        elif mode == 'fc':
            n_now = layer['n_now']
            n_prev = layer['n_prev']
            layer['W']=(np.random.rand(n_now, n_prev) - 0.5) * 0.2 # random sample in [-0.1, 0.1]
            layer['b']=(np.random.rand(n_now,1) - 0.5) * 0.2
            layer['dW']=np.zeros_like(layer['W'])
            layer['db']=np.zeros_like(layer['b'])
        #Convolution layer
        elif mode == 'conv':
            f = layer['filters']
            n_C = layer['n_C']
            n_C_prev = layer['n_C_prev']
            layer['W']=(np.random.rand(f, f, n_C_prev, n_C) - 0.5) * 0.2 # random sample in [-0.1, 0.1]
            layer['b']=(np.random.rand(1, 1, 1, n_C) - 0.5) * 0.2
            layer['dW']=np.zeros_like(layer['W'])
            layer['db']=np.zeros_like(layer['b'])
        else:
            print('Wrong layer in [{}]'.format(i))
        new_layers.append(layer)

    return new_layers
```

$S(Z) = \frac{1}{1 + e^{-Z}}$ τιμή της συνάρτησης όπου Z είναι η είσοδος της συνάρτησης. Επίσης κρατάει την τιμή του Z στην μεταβλητή cache για να χρησιμοποιηθεί αργότερα στην ανάστροφη μετάδοση (backpropagation).

2.3.3 Συνάρτηση ενεργοποίησης RELU

```
def relu_activation(Z):
    '''
    Calculates the value of relu function
```

Ο σκοπός της συνάρτησης relu_activation(Z) είναι να υπολογίσει και να επιστρέψει την τιμή της συνάρτησης $f(Z) = max(0, Z)$ όπου Z είναι η είσοδος της συνάρτησης. Πιο συγκεκριμένα επιστρέφει 0 αν η τιμή του Z είναι μικρότερη του μηδενός ή την τιμή του Z αν η τιμή είναι μεγαλύτερη του μηδενός. Επίσης κρατάει την τιμή του Z στην μεταβλητή cache για να χρησιμοποιηθεί αργότερα στην ανάστροφη μετάδοση (backpropagation).

2.3.4 Συνάρτηση ενεργοποίησης Softmax

```
def softmax_activation(Z):
    '''
    Calculates the value of Softmax turn logits (numeric output of the last linear
    layer of a multi-class classification neural network) into probabilities by taking
    the exponents of each output and then normalize each number by the sum of those
    exponents so the entire output vector adds up to one

    Arguments:
    Z -- A numeric value

    Returns:
    A -- The result of the sigmoid function calculation for the Z
    cache -- The initial value of Z
    '''
    # Softmax activation function
    n, m = Z.shape
    A = np.exp(Z)
    A_sum = np.sum(A, axis = 0)
    A_sum = A_sum.reshape(-1, m)
    A = A / A_sum
    cache = Z
    return A, cache
```

Ο σκοπός της συνάρτησης softmax_activation(Z) είναι να υπολογίσει και να επιστρέψει την τιμή της συνάρτησης

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

όπου Z είναι η είσοδος της συνάρτησης. Πιο συγκεκριμένα μετατρέπει την είσοδο από αριθμητικές τιμές σε πιθανότητες που έχουν ως άθροισμα την μονάδα. Η έξοδος της συνάρτησης είναι μια λίστα που αναπαριστά την κατανομή πιθανοτήτων μιας λίστας δυνατών αποτελεσμάτων. Επίσης κρατάει την τιμή του Z στην μεταβλητή cache για να χρησιμοποιηθεί αργότερα στην ανάστροφη μετάδοση (backpropagation).

2.3.5 Συνάρτηση επέκτασης εικόνων Zero Padding

```python
def zero_padding(X, pad):
    '''
    Pad with zeros all images of the dataset X. The padding is applied to the height and width of an image

    Arguments:
    X : numpy array of shape (m, n_H, n_W, n_C) representing a batch of m images
    pad : integer, amount of padding around each image on vertical and horizontal dimensions
    Returns:
    padded -- padded image of shape (m, n_H + 2*pad, n_W + 2*pad, n_C)
    '''

    padded = np.pad(X,((0, 0), (pad, pad), (pad, pad), (0, 0)), 'constant', constant_values=(0, 0))

    return padded
```

Ο σκοπός της συνάρτησης zero_padding(X, pad) είναι να επεκτείνει με μηδενικά κάθε εικόνα από ένα σύνολο εικόνων κατά πλάτος και κατά ύψος. Το τελικό μέγεθος της εικόνας θα είναι (n_H + 2 * pad, n_W + 2 * pad, n_C) όπου n_H είναι το ύψος της εικόνας, n_W είναι το πλάτος της εικόνας, n_C είναι ο αριθμός των καναλιών της εικόνας και pad είναι μια παράμετρος που ορίζει κατά πόσο θα επεκταθεί η εικόνα. Επιστρέφει το σύνολο από τις επεκταμένες εικόνες.

2.3.6 Συνάρτηση πρόσθιας γραμμικής ενεργοποίησης

```python
def forward_convolution(A_prev, layer):
    """
    Implements the forward propagation for a convolution function

    Arguments:
    A_prev -- output activations of the previous layer, numpy array of
    shape (m, n_H_prev, n_W_prev, n_C_prev)
```

Ο σκοπός της συνάρτησης forward_linear_activation(A_prev, layer, activation='relu') είναι να εκτελέσει την πρόσθια γραμμική ενεργοποίηση στο δοσμένο επίπεδο βάση του τύπου της ενεργοποίησης. Πιο συγκεκριμένα υπολογίζει την τιμή του Z που είναι το γινόμενο του W και του A_prev αυξημένο κατά b. Επίσης υπολογίζει την τιμή του A που είναι το αποτέλεσμα της γραμμικής ενεργοποίησης.

2.3.7 Συνάρτηση πρόσθιας συνέλιξης Forward Convolution

```
def forward_linear_activation(A_prev, layer, activation='relu'):
    '''
    Execute a linear forward activation based on the input activation type

    Arguments:
    A_prev -- The values of the previous layer
    layer -- they current layer with the hyperparameters
    activation -- the type of activation to execute on the layer

    Returns:
    A -- The results of the linear activation
    Z --  The results of the dot product of W and A_prev increased by b
    '''
    W = layer['W']
    b = layer['b']
    if activation=='sigmoid':
        Z, linear_cache=np.dot(W, A_prev)+b, (A_prev, W, b)
        A, activation_cache=sigmoid_activation(Z)
    elif activation=='relu':
        Z, linear_cache=np.dot(W, A_prev)+b, (A_prev, W, b)
        A, activation_cache=relu_activation(Z)
    else:
        Z = np.dot(W, A_prev)+b
        A = Z
    return A, Z
```

Ο σκοπός της συνάρτησης forward_convolution(A_prev, layer) είναι να υπολογίσει την πρόσθια συνέλιξη στο δοσμένο επίπεδο. Πιο συγκεκριμένα ανακτεί τις τιμές των βαρέων (weigts) και των κλίσεων (biases)καθώς και κάποιες παραμέτρους όπως το βήμα(stride)και την επέκταση (pad). Αντιστοιχούνται οι τιμές των διαστάσεων του προηγούμενου επιπέδου στις τιμές των m που είναι το πλήθος των εικόνων, των n_H_prev που είναι το

ύψος των εικόνων του προηγούμενου επιπέδου , των n_W_prev που είναι το πλάτος των εικόνων του προηγούμενου επιπέδου και του n_C_prev που είναι το πλήθος των καναλιών των εικόνων του προηγούμενου επιπέδου.

Στη συνέχεια με βάση τις υπολογισμένες τιμές του προηγούμενου επιπέδου υπολογίζεται το n_H το ύψοςκαι n_W το πλάτος του τρέχοντος επιπέδου.

Το ύψος υπολογίζεται από την εξής συνάρτηση:

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

Το πλάτος υπολογίζεται από την εξής συνάρτηση:

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

Επιπροσθέτως δημιουργείτε ένας πίνακας από μηδενικά που έχει το μέγεθος των υπολογισμένων παραμέτρων για να αποθηκεύσει το αποτέλεσμα της συνέλιξης. Το επόμενο βήμα είναι η εκτέλεση της πράξης της συνέλιξης. Η πράξη της συνέλιξης αποτελείτε από τέσσερις επαναληπτικές διαδικασίες. Σε κάθε εικόνα του δοσμένου σετ θα διαπεραστεί μία μάσκα διαστάσεων (f,f) με βήμα stride και θα υπολογιστεί η τιμή της συνέλιξης ως το άθροισμα των γινομένων των τιμών με τα βάρη (weights) αυξημένα κατά τις κλίσεις (biases).

Τέλος επιστρέφετε ο πίνακας με τα αποτελέσματα της πράξης της συνέλίξης.

2.3.8 Πρόσθια συνάρτηση εξαγωγής Forward Pooling

```
def forward_pooling(A_prev, hparameters, mode = "max"):
    '''
    Implements the forward pass of the pooling layer

    Arguments:
    A_prev -- Input data, numpy array of shape (m, n_H_prev, n_W_prev, n_C_prev)
    hparameters -- python dictionary containing "filters" and "stride"
    mode -- the pooling mode you would like to use, defined as a string ("max" or "average")

    Returns:
    A -- output of the pool layer, a numpy array of shape (m, n_H, n_W, n_C)
    cache -- cache used in the backward pass of the pooling layer, contains the input and hparameters
    '''

    # Retrieve dimensions from the input shape
    (m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape

    f = hparameters["filters"]
    stride = hparameters["stride"]

    # Define the dimensions of the output
    n_H = int(1 + (n_H_prev - f) / stride)
    n_W = int(1 + (n_W_prev - f) / stride)
    n_C = n_C_prev

    A = np.zeros((m, n_H, n_W, n_C))

    for i in range(m):                          # loop over the training examples
        for h in range(n_H):                    # loop on the vertical axis of the output volume
            # Find the vertical start and end of the current "slice"
            vert_start = stride * h
            vert_end = vert_start + f

            for w in range(n_H):                # loop on the horizontal axis of the output volume
                # Find the vertical start and end of the current "slice"
                horiz_start = stride * w
                horiz_end = horiz_start + f

                for c in range (n_C):           # loop over the channels of the output volume

                    # Use the corners to define the current slice on the ith training example of A_prev,
                    #  channel c.
                    a_prev_slice = A_prev[i,vert_start:vert_end, horiz_start:horiz_end, c]

                    # Compute the pooling operation on the slice.
                    if mode == "max":
                        A[i, h, w, c] = np.max(a_prev_slice)
                    elif mode == "average":
                        A[i, h, w, c] = np.mean(a_prev_slice)


    # Store the input and hparameters in "cache" for backward_pooling()
    cache = (A_prev, hparameters)

    return A, cache
```

Ο σκοπός της συνάρτησης forward_pooling(A_prev, hparameters, mode = “max”) είναι να υπολογίσει την πρόσθια εξαγωγή στο δοσμένο επίπεδο. Πιο συγκεκριμένα ανακτεί παραμέτρους όπως το βήμα(stride)και το μέγεθος του φίλτρου (filters). Αντιστοιχούνται οι τιμές των διαστάσεων του προηγούμενου επιπέδου στις τιμές των m που είναι το πλήθος των εικόνων, των n_H_prev που είναι το ύψος των εικόνων του προηγούμενου επιπέδου , των n_W_prev που είναι το πλάτος των εικόνων του προηγούμενου επιπέδου και του n_C_prev που είναι το πλήθος των καναλιών των εικόνων του προηγούμενου επιπέδου.
Στη συνέχεια με βάση τις υπολογισμένες τιμές του προηγούμενου επιπέδου υπολογίζεται το n_H το ύψοςκαι n_W το πλάτος του τρέχοντος επιπέδου. Επιπροσθέτως δημιουργείτε ένας πίνακας από μηδενικά που έχει το μέγεθος των υπολογισμένων παραμέτρων για να αποθηκεύσει το αποτέλεσμα της εξαγωγής. Το επόμενο βήμα είναι η εκτέλεση της πράξης της εξαγωγής. Η πράξη της εξαγωγής αποτελείτε από τέσσερις επαναληπτικές διαδικασίες. Σε κάθε εικόνα του δοσμένου σετ θα διαπεραστεί μία μάσκα διαστάσεων (f,f) με βήμα stride και θα υπολογιστεί η τιμή της εξαγωγής ως το μέγιστο ή ο μέσος όρος των τιμών βάση του δοσμένου τύπου εξαγωγής(mode).
Τέλος επιστρέφετε ο πίνακας με τα αποτελέσματα της πράξης της εξαγωγής.

2.3.9 Συνάρτηση πρόσθιας διάδοσης Forward Propagation

```
def forward_propogation(X, layers):
    '''
    Executes the forward propagation on the given layers

    Arguments:
    X -- An array containing the set of images
    layers -- An array of dictionaries where each dictionary represents a layer
```

Ο σκοπός της συνάρτησης forward_propagation(X, layers) είναι να εκτελέσει την διαδικασία της πρόσθιας μετάδοσης για κάθε ένα από τα δοσμένα επίπεδα (layers) στο

σετ εικόνων X. Αρχικά αποθηκεύετε το πλήθος των εικόνων. Στη συνέχεια γίνεται μια επαναληπτική διαδικασία στην οποία για κάθε ένα από τα επίπεδα στην λίστα layers πραγματοποιείται η αντίστοιχη πράξη.

Στην περίπτωση που το layer είναι τύπου conv δηλαδή συνελικτικό επίπεδο η τιμή του αποτελέσματος της πράξης της συνέλιξης αποθηκεύετε στην παράμετρο Z του layer και αφού εκτελεστεί και η πράξη της ενεργοποίησης RELU το η έξοδος του επιπέδου αποθηκεύετε στην παράμετρο A θα χρησιμοποιηθεί στην συνέχεια από το επόμενο επίπεδο.

Στην περίπτωση που το επίπεδο είναι τύπου pool δηλαδή επίπεδο εξαγωγής η τιμή της μεταβλητής A που είναι η έξοδος από το προηγούμενο επίπεδο αποθηκεύετε στο επίπεδο ως A_prev. Στη συνέχεια υπολογίζεται το αποτέλεσμα της πράξης της εξαγωγής κατά μέσο όρο και ενημερώνετε η μεταβλητή A η οποία είναι η έξοδος από το τρέχων επίπεδο.

Στην περίπτωση που είναι ένα πλήρως συνδεδεμένο επίπεδο αρχικά κάνουμε εξομάλυνση της λίστας(flatten) δηλαδή περνάνε όλες οι τιμές των υπολιστών στην αρχική λίστα που θα έχει γίνει μονοδιάστατη. Στην συνέχεια αποθηκεύετε η εξομαλυμένη λίστα στην παράμετρο A_prev για να κρατήσουμε την έξοδο από το προηγούμενο επίπεδο. Τέλος γίνεται η πράξη της γραμμικής πρόσθιας ενεργοποίησης. Οι έξοδοι της πράξης της πρόσθιας ενεργοποίησης που είναι το το αποτέλεσμα της ενεργοποίησης RELU ή Sigmoid αποθηκεύονται στην μεταβλητή A και στην μεταβλητή Z θα αποθηκευτούν οι τιμές των γινομένων των βαρέων (weights) με την έξοδο από το προηγούμενο επίπεδο αυξημένοι κατά τις κλίσεις (biases).

Τέλος στο τελευταίο πλήρως συνδεδεμένο επίπεδο Softmax θα αποθηκεύετε η έξοδος του προηγούμενου επιπέδου στην παράμετρο A_prev. Επίσης υπολογίζεται η πρόσθια γραμμική ενεργοποίηση και αποθηκεύετε στην μεταβλητή Z. Τέλος χρησιμοποιώντας την τιμή της μεταβλητής Z θα υπολογιστεί η έξοδος του συστήματος με βάση την πρόσθια γραμμική ενεργοποίηση Softmax που θα επιστρέψει την κατανομή των πιθανοτήτων για κάθε δυνατό αποτέλεσμα.

Επιστρέφονται η μεταβλητή A που έχει αποθηκευμένη την κατανομή των πιθανοτήτων για κάθε δυνατό αποτέλεσμα, η μεταβλητή layers που έχει αποθηκευμένες όλες τις παραμέτρων του κάθε επιπέδου(βάρη, κλίσεις). Επίσης για κάθε επίπεδο έχουν αποθηκευτεί στην μεταβλητή shapes οι διαστάσεις της κάθε εξόδου οι οποίες επίσης επιστρέφονται.

2.3.10 Υπολογισμός ακρίβειας

```
def calculate_accuracy(AL, Y):
    '''
    Measures the performance of the results of a forward_propagation.

    Arguments:
    AL -- The predicted values
    Y -- The actual values
```

Ο σκοπός της συνάρτησης calculate_accuracy(AL, Y) είναι τα υπολογίσει την απόδοση του δικτύου. Λαμβάνει ως είσοδο τις τιμές που εκτιμήθηκαν από το μοντέλο και τις πραγματικές τιμές. Επιστρέφει την ακρίβεια του δικτύου που είναι το πηλίκο της διαίρεσης του πλήθους των επιτυχημένων εκτιμήσεων με τον αριθμό των εκτιμήσεων.

2.3.11 Υπολογισμός κόστους

```
def calculate_cost(AL, Y):
    '''
    Calculates the cost of the network

    Arguments:
    AL -- The predicted values
    Y -- The actual values

    Returns:
    cost -- The cost of the network

    '''
    n, m = Y.shape
    cost = - np.sum(np.log(AL) * Y) / m
    cost=np.squeeze(cost)

    return cost
```

Ο σκοπός της συνάρτησης calculate_cost(AL, Y) είναι να υπολογίσει το πόσο μη αποδοτικόείναι το δίκτυο. Πιο συγκριμένα δίνονται ως είσοδοι οι εκτιμημένες τιμές και οι πραγματικές τιμές και βάση των λανθασμένων εκτιμήσεων υπολογίζεται το κόστος.Η συνάρτηση χρησιμοποιείτε από τη συνάρτηση ανάστροφηςμετάδοσης(Back Propagation).

2.3.12 Μέθοδος ανάστροφης συνέλιξης Backward Convolution

```
def backward_convolution(dZ, layer):
    """
    Implement the backward propagation for a convolution function

    Arguments:
    dZ -- gradient of the cost with respect to the output of the conv layer (Z),
    numpy array of shape (m, n_H, n_W, n_C)
```

Ο σκοπός της συνάρτησης backward_convolution(A_prev, layer) είναι να υπολογίσει την ανάστροφη συνέλιξη στο δοσμένο επίπεδο. Πιο συγκεκριμένα ανακτεί τις τιμές των A_prev που είναι η έξοδος από το προηγούμενο επίπεδο των βαρέων (weigts) και των κλίσεων (biases)καθώς των Z που είναι οι τιμές που είναι το αποτέλεσμα της συνέλιξης και κάποιες παραμέτρους όπως το βήμα(stride)και την επέκταση (pad). Αντιστοιχούνται οι τιμές των διαστάσεων του προηγούμενου επιπέδου στις τιμές των m που είναι το πλήθος των εικόνων, των n_H_prev που είναι το ύψος των εικόνων του προηγούμενου επιπέδου , των n_W_prev που είναι το πλάτος των εικόνων του προηγούμενου επιπέδου και του n_C_prev που είναι το πλήθος των καναλιών των εικόνων του προηγούμενου επιπέδου. Η παραπάνω διαδικασία επαναλαμβάνεται και για τις διαστάσεις των βαρέων (weights) και των Z.

Στη συνέχειαδημιουργούνται τρεις πίνακες από μηδενικά από το μέγεθος των υπολογισμένων παραμέτρων για να αποθηκευτούν οι πίνακες για διαφορικά dA_prev, dW και db ως προς το κόστος.

Το διαφορικό της εξόδου του προηγούμενου επιπέδου υπολογίζεται από τον εξής τύπο:

$$dA+= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_c \times dZ_{hw}$$

Το διαφορικό του βάρους (weight) ως προς το κόστος υπολογίζεται από τον εξής τύπο:

$$dW_c+= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} \times dZ_{hw}$$

Το διαφορικό της κλίσης (bias) ως προς το κόστος υπολογίζεται από τον εξής τύπο:

$$db = \sum_{h} \sum_{w} dZ_{hw}$$

Το επόμενο βήμα είναι η εκτέλεση της πράξης της ανάστροφης συνέλιξης. Η πράξη της ανάστροφης συνέλιξης όπως και της πρόσθιας αποτελείτε από τέσσερις επαναληπτικές διαδικασίες. Σε κάθε εικόνα του δοσμένου σετ θα διαπεραστεί μία μάσκα διαστάσεων (f,f) με βήμα stride και θα υπολογιστούν οι τιμές των διαφορικών.

Επιπροσθέτως ελέγχεται αν έχει γίνει επέκταση (padding) στο εκάστοτε επίπεδο και στην περίπτωση που έχει γίνει αναστρέφετε η επέκταση (padding).

Τέλος επιστρέφονται οι τιμές των διαφορικών dA_prev, dW και db.

2.3.13 Συνάρτηση δημιουργίας μάσκας από πίνακα

```
def create_mask_from_matrix(x):
    '''
    Creates a mask from an input matrix x, to identify the max entry of x.

    Arguments:
    x : Array of shape (f, f)
```

Ο σκοπός της συνάρτησης create_mask_from_matrix(x) είναι να δημιουργήσει μια μάσκα από έναν πίνακα x. Πιο συγκεκριμένα η συνάρτηση επιστρέφει έναν πίνακα ίδιου μεγέθους με τον x που περιέχει True στις θέσεις πουέχουν τιμή ίση με την μέγιστη τιμή του x.

2.3.14 Συνάρτηση καταμερισμού τιμών σε πίνακα

```
def distribute_values_into_matrix(dz, shape):
    '''
    Distributes the input value in the matrix of dimension shape

    Arguments:
    dz : input scalar
    shape : the shape (n_H, n_W) of the output matrix for which we want to
    distribute the value of dz

    Returns:
    a : Array of size (n_H, n_W) for which we distributed the value of dz
    '''

    # Retrieve dimensions from shape
    (n_H, n_W) = shape

    # Compute the value to distribute on the matrix
    average = np.ones([n_H, n_W]) / (n_H * n_W)

    # Create a matrix where every entry is the "average" value
    a = dz * average

    return a
```

Ο σκοπός της συνάρτησης distribute_values_into_matrix(x) είναι να τοποθετήσει τις δοσμένες τιμές dZ σε έναν πίνακα διαστάσεων shape. Επιστρέφει έναν πίνακα α διαστάσεων shape που περιέχει τις μέσες τιμές που το άθροισμα τους δίνει την τιμή του dZ.

2.3.15 Ανάστροφη συνάρτηση εξαγωγής Backward Pooling

```
def backward_pooling(dA, layer, mode = "max"):
    """
    Implements the backward pass of the pooling layer

    Arguments:
    dA -- gradient of cost with respect to the output of the pooling layer, same shape as A
    layer -- the pooling layer, contains the layer's input and hparameters
    mode -- the pooling mode you would like to use, defined as a string ("max" or "average")
```

Ο σκοπός της συνάρτησης backward_pooling(dA, layer, mode = “max”) είναι να υπολογίσει την ανάστροφη εξαγωγή στο δοσμένο επίπεδο. Πιο συγκεκριμένα ανακτεί τις τιμές των A_prev που είναι η έξοδος από το προηγούμενο επίπεδο, το βήμα(stride)και το μέγεθος του φίλτρου (filters). Αντιστοιχούνται οι τιμές των διαστάσεων του προηγούμενου επιπέδου στις τιμές των m που είναι το πλήθος των εικόνων, των n_H_prev που είναι το ύψος των εικόνων του προηγούμενου επιπέδου , των n_W_prev που είναι το πλάτος των εικόνων του προηγούμενου επιπέδου και του n_C_prev που είναι το πλήθος των καναλιών των εικόνων του προηγούμενου επιπέδου. Επίσης αντιστοιχούνται οι τιμές των διαστάσεων του διαφορικού του A που είναι η έξοδος του τρέχοντος επιπέδου.

Στη συνέχεια δημιουργείτε ένας πίνακας από μηδενικά που έχει το μέγεθος του A_prev για να αποθηκεύσει το αποτέλεσμα της ανάστροφης εξαγωγής. Το επόμενο βήμα είναι η εκτέλεση της πράξης της ανάστροφης εξαγωγής. Η πράξη της εξαγωγής αποτελείτε από τέσσερις επαναληπτικές διαδικασίες. Σε κάθε εικόνα του δοσμένου σετ θα διαπεραστεί μία μάσκα διαστάσεων (f,f) με βήμα stride και θα υπολογιστεί η τιμή της ανάστροφης εξαγωγής ως το μέγιστο ή ο μέσος όρος των τιμών βάση του δοσμένου τύπου εξαγωγής(mode).

Τέλος επιστρέφετε το διαφορικό του A_prev.

2.3.16 Ανάστροφη συνάρτηση ενεργοποίησης Sigmoid

```
def backward_sigmoid(dA, cache):
    '''
    Calculates the backward propagation of sigmoid activation
    function

    Arguments:
    dA -- gradient of cost with respect to the input of the
    conv layer
    cache -- The cached output of the convolution

    Returns:
    dZ -- gradient of cost with respect to the output of the
    conv layer
    '''
    # Backpropogation of sigmoid activation function
    Z = cache
    s = 1/(1+np.exp(-Z))
    dZ = dA * s * (1-s)
    return dZ
```

```
def backward_relu(dA, cache):
    '''
    Calculates the backward propagation of RELU activation
    function

    Arguments:
    dA -- gradient of cost with respect to the input of the
    conv layer
    cache -- The cached output of the convolution

    Returns:
    dZ -- gradient of cost with respect to the output of the
    conv layer
    '''
    # Backpropogation of Relu activation function
    Z = cache
    dZ = np.array(dA, copy=True) # just converting dz to a correct object.
    dZ[Z < 0] = 0
    return dZ
```

Ο σκοπός της συνάρτησης backward_sigmoid(dA, cache) είναι να υπολογίσει την ανάστροφη τιμή της συνάρτησης ενεργοποίησης Sigmoid. Πιο συγκεκριμένα λαμβάνει σαν είσοδο την τιμή του διαφορικού του A και την cache δηλαδή την αποθηκευμένη τιμή του Z. Στην συνέχεια υπολογίζεται η τιμή της ανάστροφης συνάρτησης ενεργοποίησης Sigmoid.

2.3.17 Ανάστροφη συνάρτηση ενεργοποίησης RELU

Ο σκοπός της συνάρτησης backward_relu(dA, cache) είναι να υπολογίσει την ανάστροφη τιμή της συνάρτησης ενεργοποίησης RELU. Πιο συγκεκριμένα λαμβάνει σαν είσοδο την τιμή του διαφορικού του A και την cache δηλαδή την αποθηκευμένη τιμή του Z. Στην συνέχεια υπολογίζεται η τιμή της ανάστροφης συνάρτησης ενεργοποίησης RELU.

2.3.18 Ανάστροφη συνάρτηση ενεργοποίησης Softmax

```
def backward_softmax(A, Y):
    '''
    Calculates the backward propagation of sigmoid activation
    function

    Arguments:
    A -- The input of the fully connected layer
    Y -- The cached output of the fully connected layer

    Returns:
    dZ -- gradient of cost with respect to the output of the
```

Ο σκοπός της συνάρτησης backward_softmax(A, Y) είναι να υπολογίσει την ανάστροφη τιμή της συνάρτησης ενεργοποίησης Softmax. Πιο συγκεκριμένα λαμβάνει σαν είσοδο την τιμή του A και την Y δηλαδή την αποθηκευμένη τιμή της εξόδου του πλήρως συνδεδεμένου επιπέδου. Στην συνέχεια υπολογίζεται η τιμή της ανάστροφης συνάρτησης ενεργοποίησης Softmax.

2.3.19 Ανάστροφη συνάρτηση γραμμικής ενεργοποίησης

```
def backward_linear_activation(dA, layer, activation):
    '''
    Calculates the backward propagation of a linear activation
    function

    Arguments:
    dA -- The gradient of cost with respect to the input of the
    conv layer
    layer -- Dictionary containing the layer parameters
    activation -- The type of the activation

    Returns:
    dA_prev -- The gradient of cost with respect to the input of
    the previous layer
    dW -- The gradient of the cost with respect to the weight
    db -- The gradient of the cost with respect to the biases
    '''
    # Backward propagatIon module - linear activation backward
    A_prev = layer['A_prev']
    W = layer['W']
    b = layer['b']
    Z = layer['Z']
    if activation=='relu':
        dZ=backward_relu(dA, Z)
    elif activation=='sigmoid':
        dZ=backward_sigmoid(dA, Z)
    else:
        dZ = dA
    n, m = dA.shape
    dA_prev=np.dot(W.T, dZ)
    dW = np.dot(dZ, A_prev.T)
    db = np.sum(dZ, axis = 1).reshape(n,1)

    return dA_prev, dW, db
```

Ο σκοπός της συνάρτησης backward_linear_activation(dA, layer, activation) είναι να εκτελέσει την ανάστροφη γραμμική ενεργοποίηση. Πιο συγκεκριμένα λαμβάνει ως είσοδο τις παραμέτρους dA που είναι το διαφορικό του A, το layer που έχει αποθηκευμένες τις παραμέτρους του επιπέδου καθώς και τον τύπο της ενεργοποίησης activation. Στην συνέχεια εξάγονται οι τιμές από το layer και αποθηκεύονται στο W (βάρη),b (κλίσεις) και Z η έξοδος του επιπέδου. Έπειτα εκτελείτε η εκάστοτε πράξη βάση τον τύπο της ενεργοποίησης. Τέλος υπολογίζονται οι τιμές των dA_prev dW και db.

2.3.20 Συνάρτηση ανάστροφης μετάδοσης Backward Propagation

```
def backward_propagation(AL, Y, layers):
    '''
    Executes the backward propagation for all layers

    Arguments:
    AL -- The input of the fully connected layer
    Y -- The cached output of the fully connected layer
    layers -- A list of all layers with the respective parameters

    Returns:
    layers -- A list of all layers with the respective updated parameters
    '''
    m = Y.shape[1]

    # Last fully connected softmax layer
    dZ = backward_softmax(AL, Y)
    dA_prev, dW, db = backward_linear_activation(dZ, layers[-1], 'none')
    layers[-1]['dW'] = dW
    layers[-1]['db'] = db

    for layer in reversed(layers[:-1]):
        flattend = True
        if layer['mode'] == 'fc':
            dA_prev, dW, db = backward_linear_activation(dA_prev, layer, 'relu')
            layer['dW'] = dW
            layer['db'] = db
        elif layer['mode'] == 'conv':
            if flattend:
                dA = (dA_prev.T).reshape(m,1,1,layer['n_C']) # flatten backward
                flattend = False

            dZ = backward_relu(dA, layer['Z'])
            dA_prev, dW, db = backward_convolution(dZ, layer)
            layers['dW'] = dW
```

Ο σκοπός της συνάρτησης backward_propagation(AL, Y, layers) είναι να εκτελέσει την ανάστροφη μετάδοση για κάθε επίπεδο. Πιο συγκεκριμένα λαμβάνει ως είσοδο AL που είναι η είσοδος του τελευταίου πλήρως συνδεδεμένου Softmax επιπέδου, το Y που είναι η έξοδος του του τελευταίου πλήρως συνδεδεμένου Softmax επιπέδου δηλαδή η κατανομή των πιθανοτήτων για κάθε ένα από τα πιθανά αποτελέσματα και η λίστα layers που είναι η λίστα με τα αποθηκευμένα επίπεδα καθώς και τις παραμέτρους για κάθε επίπεδο. Για κάθε ένα από τα επίπεδα ξεκινώντας από το τελευταίο εκτελεί την εκάστοτε ανάστροφη πράξη και ενημερώνει τις αντίστοιχες τιμές. Τέλος επιστρέφει την ενημερωμένη λίστα των επιπέδων.

2.3.21 Συνάρτηση ενημέρωσης παραμέτρων

```
def update_parameters(layers, learning_rate):
    '''
    Updates the parameters of the layer

    Arguments:
    layers -- The list with the layers along with the coresponding hyperparameters
    learning_rate -- The learning rate by which the model is learning

    Returns:
    layers -- The updated list with the layers along with the oresponding hyperparameters
    '''
    num_layer = len(layers)
    for i in range(num_layer):
        mode = layers[i]['mode'] # 'fc', 'conv', 'pool'
        if mode == 'pool':
            continue
        elif (mode == 'fc' or mode == 'conv'):
            layers[i]['W'] = layers[i]['W'] - learning_rate*layers[i]['dW']
            layers[i]['b'] = layers[i]['b'] - learning_rate*layers[i]['db']
        else:
            print('Wrong layer mode in [{}]'.format(i))

    return layers
```

Ο σκοπός της συνάρτησης update_parameters(layers, learning_rate) είναι να ενημέρωση τις τιμές της λίστας των επιπέδων. Πιο συγκεκριμένα λαμβάνει ως είσοδο την λίστα με τα επίπεδα και τις παραμέτρους του κάθε επιπέδου και τον ρυθμό μάθησης learning_rate που αντιστοιχεί στο κατά πόσο ενημερώνονται τα βάρη (weights)και οι κλίσεις (biases)σε κάθε επανάληψη. Στην περίπτωση που το επίπεδο είναι πλήρως συνδεδεμένο επίπεδο καθώς και στην περίπτωση που είναι συνελικτικό επίπεδο η ενημέρωση των τιμών των βαρέων και των κλίσεων είναι ανάλογη του ρυθμού μάθησης του δικτύου. Τέλος επιστρέφεται η λίστα των επιπέδων με τις ενημερωμένες τιμές.

2.3.22 Συνάρτηση κατηγοριοποίησης Predict

```
def predict(X_test, Y_test, layers):
    '''
    Makes a prediction based on the trained layers and calculates the accuracy
    of the prediction

    Arguments:
    X_test -- The dataset with the images
    Y_test -- The actual labels of the images
    layers -- The trained layers

    Returns:
    pred -- The predicted labels
    accuracy -- The accuracy of the prediction
    '''

    m = X_test.shape[0]
    n = Y_test.shape[1]
    pred = np.zeros((n,m))
    pred_count = np.zeros((n,m)) - 1 # for counting accurate predictions

    # Forward propagation
    AL, _, _ = forward_propogation(X_test, layers)

    # convert prediction to 0/1 form
    max_index = np.argmax(AL, axis = 0)
    pred[max_index, list(range(m))] = 1
    pred_count[max_index, list(range(m))] = 1

    accuracy = np.float(np.sum(pred_count == Y_test.T)) / m

    return pred, accuracy
```

Ο σκοπός της συνάρτησης predict(X_test, Y_test, layers) είναι να κάνει την κατηγοριοποίηση βάση του εκπαιδευμένου μοντέλου. Πιο συγκεκριμένα λαμβάνει σαν είσοδο το σετ των εικόνων X_test για τις οποίες θα γίνει η κατηγοριοποίηση, μια λίσταY_test που περιέχει τις πραγματικές κατηγορίες των εικόνων καθώς και την λίστα των εκπαιδευμένων επιπέδων layers. Στη συνέχεια τροφοδοτούνται η εικόνες στο μοντέλο μέσω της συνάρτησης πρόσθιας διάδοσης (forward propagation) και υπολογίζεται η ακρίβεια της κατηγοριοποίησης με βάση τις επιτυχημένες ταξινομήσεις. Τέλος επιστρέφονται οι λίστα με τις εκτιμημένες κατηγορίες καθώς και η ακρίβεια του μοντέλου.

2.3.23 Συνάρτηση εκπαίδευσης Train

```
def train(X_train, Y_train, X_test, Y_test, layers, batch_size=10, num_epoch=1, learning_rate=0.01):
    '''
    Trains the modeles using the predifined functions

    Arguments:
    X_train -- The set of train images
    Y_train -- The actuall labels of train images
    X_test -- The set of test images
```

Η τελευταία συνάρτηση της βιβλιοθήκης είναι η train(X_train, Y_train, X_test, Y_test, layers, batch_size=10, num_epoch=1, learning_rate=0.01)η οποία συνδέει όλες τις παραπάνω συναρτήσεις για να εκπαιδεύσει το δοσμένο μοντέλο.

```
        accuracy_test_list.append(accuracy_test)
        print('Epoch [{}] average_loss = {} average_accuracy = {}'.format(epoch, np.mean(losses), np.mean(accuracies)))
        print('Epoch [{}] train_loss = {} train_accuracy = {}'.format(epoch, loss_train, accuracy_train))
        print('Epoch [{}] test_accuracy = {}'.format(epoch, accuracy_test))

    return layers, accuracy_train_list, accuracy_test_list
```

Πιο συγκεκριμένα λαμβάνει ως είσοδο τα σετ των εικόνων εκπαίδευσης και επαλήθευσης καθώς και τις λίστες με τις κατηγορίες στις οποιες ανήκει κάθε εικόνα, την αρχιτεκτονική του μοντέλου δηλαδή μια λιστα με τον αριθμό των επιπεδων τον τύπο και τις παραμέτρους του κάθε επιπέδου καθώς και το batch size που είναι το πόσες εικόνες θα εκπαιδεύονται ταυτόχρονα σε μια επανάληψη, το num_epoch που είναι ο αριθμός των επαναλήψεων που θα εκτελεστούν στο set από τις εικόνες και το learning_rate δηλαδή το κατά πόσο θα ενημερώνονται οι τιμές των βαρέων και κλίσεων σε κάθε επανάληψη.

Στη συνέχεια υπολογίζεται ο αριθμός των επαναλήψεων για κάθε epoch ως το πηλίκο του πλήθους των εικόνων με το μέγεθος του batch. Ο συνολικός αριθμός των επαναλήψεων είναι ίσος με τον γινόμενο του πλήθους των epochs με τον αριθμό των επαναλήψεων για κάθε epoch. Για παράδειγμα εάν έχουμε 60000 χιλιάδες εικόνες με batch size 100 και αριθμό epoch 10 οι συνολικές επαναλήψεις θα είναι 10 * 60000 / 100 δηλαδή 6000.

Επιπροσθέτως στην πρώτη επανάληψη θα τυπωθούν οι διαστάσεις της εξόδου του κάθε επιπέδου. Επίσης για κάθε 300 επαναλήψεις μέσα σε κάθε epoch καθώς και στο τέλος του κάθε epoch θα τυπώνονται η τρέχων ακρίβεια και το τρέχων κόστος και θα αποθηκεύονται σε μια λίστα για να μπορεί να γίνει μελέτη της μεταβολής των παραπάνω παραμέτρων.

Τέλος επιστρέφονται η λίστα με τα εκπαιδευμένα επίπεδα και τις ενημερωμένες παραμέτρους καθώς και οι λίστες με τις ακρίβειες των σετ εκπαίδευσης και επαλήθευσης για κάθε epoch.

Συμπεράσματα

Υστέρα από την διεκπεραίωση των πειραμάτων και την ανάλυση των αποτελεσμάτων είναι φανερό ότι η παραλλαγή του μοντέλου VGG που εκπαιδεύτηκε χρησιμοποιώντας την βιβλιοθήκη Keras είναι πιο αποδοτική σε σχέση με το μοντέλο LeNet-5 είτε είναι αυτό εκπαιδευμένο με την βιβλιοθήκη Keras είτε με την βιβλιοθήκη που υλοποιήθηκε στα πλαίσια της πτυχιακής εργασίας. Επίσης αξίζει να σημειωθεί πως το μοντέλο LeNet-5εκπαιδευμένο με την βιβλιοθήκη Kerasεκπαιδεύετε πιο γρήγορα από το παραλλαγμένο μοντέλο VGG καθώς ο αριθμός των επιπέδων αυξάνει την διάρκεια εκπαίδευσης του μοντέλου.

Επιπροσθέτως αξίζει να επισημανθεί ότι η εκπαίδευση των μοντέλων χρησιμοποιώντας την βιβλιοθήκη που υλοποιήθηκε στα πλαίσια της πτυχιακής είναι πολλές φορές πιο χρονοβόρα. Στην σύγκριση φάνηκε ότι η διάρκεια της εκπαίδευσης του μοντέλου LeNet-5 με την βιβλιοθήκη Keras ήταν χιλιάδες φορές μικρότερη σε σχέση με την δική μας βιβλιοθήκη.

Ακόμη ένα αξιοσημείωτο γεγονός είναι αυτό της δημιουργίας μια βιβλιοθήκης για την δημιουργία συνελικτικών νευρωνικών δικτύων. Ο βαθμός δυσκολίας είναι αρκετά μεγάλος ιδιαίτερα όσον αφορά τις διαστάσεις των εικόνων. Πρέπει να γίνει σωστός χειρισμός των διαστάσεων σε κάθε επίπεδο έτσι ώστε να αποτραπούν τα σφάλματα που αφορούν τις διαστάσεις των εικόνων. Επίσης χρησιμοποιώντας την βιβλιοθήκη για την εκπαίδευση των μοντέλων συμπεραίνουμε ότι είναι αναγκαία η βελτιστοποίηση του κώδικα και να γραφτούν τα σημαντικά και χρονοβόρα σημεία σε γλώσσα χαμηλότερου επιπέδου έτσι ώστε να επιταχυνθεί η εκπαίδευση των μοντέλων.

Βιβλιογραφία

[1] Rajeev Ranjan , Carlos D. Castillo , Rama Chellappa, ”L2-constrained Softmax Loss for Discriminative Face Verification”, arXiv 2017

[2] Zeiler M.D., Fergus R., ”Visualizing and Understanding Convolutional Networks”, In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8689. Springer, Cham.

[3] Yandong Wen, Kaipeng Zhang, Zhifeng Li, Yu Qiao, ”A Discriminative Feature Learning Approach for Deep Face Recognition”, In ECCV, 2016

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, “Going deeper with convolutions “, In Cvpr, 2015

[5] Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, ”Gradient-Based Learning Applied to Document Recognition”, In Proceedings of the IEEE, Volume: 86, Issue 11, Nov 1998, Pages: 2278 - 2324

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, ”ImageNet classification with deep convolutional neural networks”, In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105. 2012.

[7] Weiyang Liu , Yandong Wen , Zhiding Yu , Meng Yang, ”Large-Margin Softmax Loss for Convolutional Neural Networks”, In ICML, 2016

[8] J. Hu, L. Shen, and G. Sun, ”Squeeze-and-excitation networks “, arXiv preprint arXiv:1709.01507, 2017.

[9] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart Russell, “Distance metric learning with application to clustering with side-information”, In Proceedings of the 15th International Conference on Neural Information Processing Systems, Pages 521-528, 2002

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, ”Going deeper with convolutions ”, In Cvpr, 2015

[11] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556:1–10, 2014. URL http://arxiv.org/abs/1409.1556.

[12] K. He, X. Zhang, S. Ren, and J. Sun, ”Deep residual learning for image recognition ”, In CVPR, pages 770– 778, 2016.

[13] Sergey Ioffe, Christian Szegedy, ”Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift ”, arXiv preprint, arXiv:1502.03167v3, 2015

[14] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, InceptionResNet and the Impact of Residual Connections on Learning. In ICLR 2016 Workshop, 2016. URL https://arxiv.org/abs/1602.07261.
[15] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. Neural Networks, 61: 85–117, 2015.

[16] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '14, pages 1717–1724. IEEE Computer Society, 2014.

[17] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the ImageNet. CoRR, abs/1606.02228, 2016. URL http://arxiv.org/abs/1606.02228.

[18] Thomas M. Mitchell. Machine Learning. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 9780070428072.

[19] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld. Adaptive Histogram Equalization and Its Variations. Computer Vision, Graphics, and Image Processing, 39(3):355–368, 1987.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385, 2015.

[21] Rich Caruana. A dozen tricks with multitask learning. In Neural Networks: Tricks of the Trade - Second Edition, Lecture Notes in Computer Science, pages 163–189. Springer, 2012.

[22] Rich Caruana. Multitask Connectionist Learning. In In Proceedings of the 1993 Connectionist Models Summer School, pages 372–379, 1993.