ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ ΓΙΑ ΤΑΞΙΝΟΜΗΣΗ ΕΙΚΟΝΩΝ ΣΠΟΡΟΦΥΤΩΝ

Διπλωματική Εργασία

του

Χρήστου Κουρούνη

Θεσσαλονίκη, Σεπτέμβριος 2019

ΣΧΕΔΙΑΣΗ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ ΓΙΑ ΤΑΞΙΝΟΜΗΣΗ ΕΙΚΟΝΩΝ ΣΠΟΡΟΦΥΤΩΝ

Χρήστος Κουρούνης

Πτυχίο Εφαρμοσμένης Πληροφορικής, ΠΑΜΑΚ, 2016

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Ρεφανίδης Ιωάννης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 01/11/2019

Ρεφανίδης Ιωάννης	Σαμαράς Νικόλαος	Σακελλαρίου Ηλίας
.....	.....	.....

Χρήστος Κουρούνης

.....

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι μια ολοκληρωμένη παρουσίαση, του τρόπου με τον οποίο δημιουργείται από την αρχή ένα νευρωνικό δίκτυο για την ταξινόμηση εικόνων.

Στα πλαίσια της διπλωματικής θα παρουσιαστεί το ελάχιστο θεωρητικό υπόβαθρο που απαιτείται, για τα άτομα που θέλουν να ασχοληθούν με το χώρο της μηχανικής μάθησης, και του υπό-πεδίου της, της βαθιάς μάθησης, κάνοντας μια μικρή περιγραφή της ιστορίας των τομέων αυτών, περιγράφοντας βασικές έννοιες και προβλήματα που παρουσιάζονται στη δημιουργία ενός νευρωνικού δικτύου και ο τρόπος με τον οποίο μπορούν να επιλυθούν.

Επίσης γίνεται αναφορά και περιγραφή του τρόπου με τον οποίο χρησιμοποιούνται κάποια από τα πιο πολυχρησιμοποιημένα εργαλεία για τη δημιουργία των νευρωνικών δικτύων, όπως το TensorFlow και το keras, στη δημιουργία ενός δικτύου για την ταξινόμηση εικόνων σε πολλαπλές κατηγορίες. Χρησιμοποιούνται έτοιμα δεδομένα από το kaggle, που αφορούν 12 διαφορετικές κατηγορίες φυτών σε διάφορα στάδια της ανάπτυξής τους.

Στο πρακτικό μέρος, γίνεται δημιουργία πολλαπλών δικτύων, διαδοχικά, όπου βάσει των αποτελεσμάτων τους, εξάγονται συμπεράσματα και αναλύονται τα προβλήματα τους, με την προσπάθεια επίλυσής τους στην επόμενη εκδοχή του δικτύου. Με αυτόν τον τρόπο δημιουργούνται όλο και πιο περίπλοκα δίκτυα είτε μέσω της χρήσης καινούριων επιπέδων είτε με την επαναληπτική χρήση των ήδη υπαρχόντων, μέχρις ότου δημιουργηθεί ένα δίκτυο με αρκετά υψηλή ακρίβεια (96,7%) ταξινόμησης.

Λέξεις Κλειδιά:

Νευρωνικά δίκτυα, μηχανική μάθηση, Tensorflow, Python, keras

Abstract

The purpose of this master thesis is to provide a comprehensive presentation on how to create a neural network for image classification from the beginning.

The thesis will present the minimum theoretical background required for those wishing to pursue a career in the field of mechanical learning and its sub-field of deep learning, giving a brief outline of the history of these areas, outlining key concepts. and problems that arise in the creation of a neural network and how they can be resolved.

It also mentions and describes how some of the most widely used neural network tools, such as TensorFlow and keras, are used to create a network to classify images into multiple categories. The data being used are from a kaggle competition, and they show 12 different categories of plants at various stages of their development.

In the practical part of the thesis, multiple networks are created in succession, where based on results of the previous network, conclusions are being drawn and an analysis of their problems is being made, in an attempt to solve them in the next version of the network. In this way, more and more complex networks are created, either through the use of new layers or the repeated use of existing ones, until a highly accurate (96.7%) classification network is created.

Keywords:

Neural networks, machine learning, TensorFlow, Python, keras

Περιεχόμενα

Κατάλογος Εικόνων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Στα πλαίσια της διπλωματικής εργασίας θα γίνει περιγραφή των βασικών θεωρητικών εννοιών που απαιτούνται για την δημιουργία ενός νευρωνικού δικτύου που θα καταφέρει να κατηγοριοποιεί εικόνες 12 διαφορετικών ειδών φυτών με αρκετά υψηλή ακρίβεια. Γίνεται χρήση της προγραμματιστικής γλώσσας Python, αλλά και επιμέρους βιβλιοθηκών όπως το TensorFlow και το keras. Θα παρουσιαστούν τα προβλήματα που συνήθως εμφανίζονται σε μια τέτοια διαδικασία, οι τεχνικές που χρησιμοποιούνται από τους περισσότερους ερευνητές για να τα επιλύσουν καθώς και η παρουσίαση τρόπων για να επιτευχθούν όσο το δυνατόν καλύτερα αποτελέσματα.

Όλο και περισσότερες τεχνολογίες της καθημερινότητάς μας χρησιμοποιούν διαδικασίες μηχανικής μάθησης για την αυτοματοποίηση διαφορετικών εργασιών που μέχρι πρόσφατα μπορούσαν να επιτευχθούν μόνο από ανθρώπους, όπως η αναγνώριση εικόνων, που έχει οδηγήσει σε αμάξια που «οδηγούν» μόνα τους. Επιχειρήσεις ανά τον κόσμο έχουν αντιληφθεί τη ζωτική σημασία που έχουν τα δεδομένα των πελατών τους και πως μπορούν να οδηγήσουν σε αυξημένες πωλήσεις, εφόσον έχουν χρησιμοποιηθεί με σωστό τρόπο.

Αυτοί οι λόγοι, και πολλοί άλλοι, σε συνδυασμό με την αυξημένη προσβασιμότητα του τομέα της μηχανικής μάθησης τα τελευταία χρόνια, έχουν δημιουργήσει τις κατάλληλες συνθήκες για τον κάθε ενδιαφερόμενο να μπορέσει να κάνει τα πρώτα του «βήματα» στον χώρο.

1.2 Σκοπός – Στόχοι

Στόχος αυτής της διπλωματικής είναι η δημιουργία ενός φροντιστηριακού κειμένου που θα κάνει μια περιγραφή του απαραίτητου θεωρητικού υπόβαθρου που απαιτείται για την ενασχόληση με τα νευρωνικά δίκτυα, ξεκινώντας από μια ιστορική αναδρομή, μέχρι και την παρουσίαση των πιο κοινών μεθόδων και τεχνικών που χρησιμοποιούνται για τη δημιουργία ενός νευρωνικού δικτύου που κάνει κατηγοριοποίηση εικόνων σε πολλαπλές κατηγορίες, από το μηδέν.

Η δημιουργία του δικτύου κάνει χρήση της γλώσσας Python και των βιβλιοθηκών TensorFlow και keras. Θα γίνει παρουσίαση της προ επεξεργασίας των δεδομένων που χρησιμοποιήθηκαν (εικόνες από 12 διαφορετικές κατηγορίες φυτών) αλλά και πλήρης περιγραφή της δημιουργίας του, από το μηδέν, εμπλουτίζοντας το

δίκτυο μετά από κάθε «τρέξιμό» του ανάλογα με τα προβλήματα που παρουσιάζονται, μέχρις ότου δημιουργηθεί ένα δίκτυο ικανό να ταξινομεί τις εικόνες με αρκετά μεγάλη ακρίβεια.

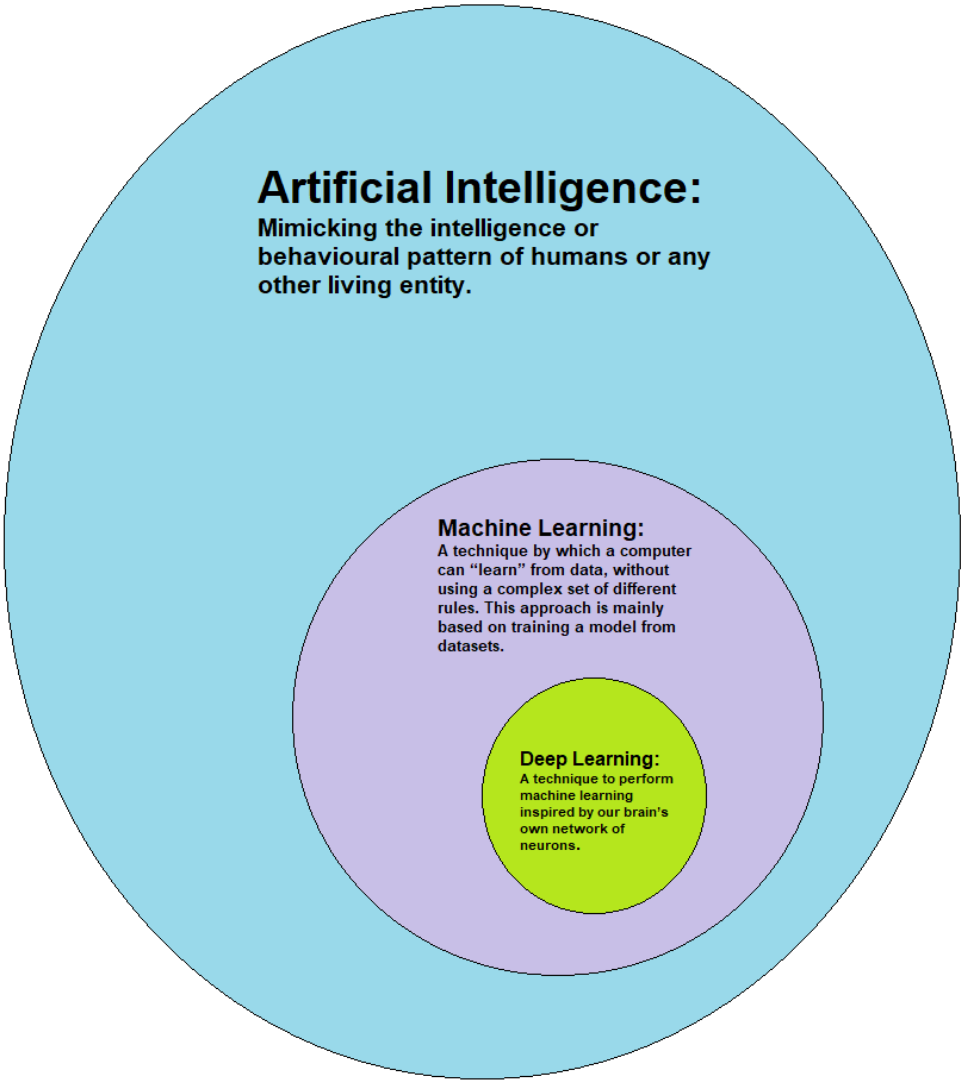
1.3 Διάρθρωση της μελέτης

Στο Κεφάλαιο 2 γίνεται περιγραφή των όρων τεχνητής νοημοσύνης, μηχανικής μάθησης και της βαθιάς μάθησης, της διαφοράς μεταξύ τους καθώς και μια ιστορική αναδρομή. Στη συνέχεια περιγράφονται οι κατηγορίες της μηχανικής μάθησης, η δομή των νευρωνικών δικτύων καθώς από που αντλήθηκε η ιδέα για τη δημιουργία τους. Γίνεται αναφορά στον τρόπο αναπαράστασης των εικόνων μέσα στη μνήμη του υπολογιστή και τον τύπο δικτύων που χρησιμοποιούν τις εικόνες ως δεδομένα εισόδου αλλά και κάποια σημεία κλειδιά των δικτύων που θα χρησιμοποιήσουμε. Τελειώνοντας το θεωρητικό κομμάτι, περιγράφεται ο τρόπος με τον οποίο θα διαχωριστούν τα δεδομένα. Στο 3ο κεφάλαιο γίνεται παρουσίαση του προβλήματος, των δυσκολιών που εμπεριέχει καθώς και των εργαλείων που χρησιμοποιούνται, δηλαδή τις βιβλιοθήκες της Python. Στο 4ο κεφάλαιο, περιγράφεται αναλυτικά η προ επεξεργασία των δεδομένων που χρησιμοποιούνται, καθώς και ο τρόπος δημιουργίας του δικτύου, από το μηδέν, μέχρι και το τελικό δίκτυο υψηλής ακρίβειας που σύμφωνα με τα αποτελέσματά του δημιουργείται το αρχείο που αξιολογεί ο διαγωνισμός του Kaggle. Στο τελευταίο κεφάλαιο (5ο), γίνεται μια σύντομη περιγραφή των αποτελεσμάτων της διπλωματικής, καθώς και μελλοντικών επεκτάσεων.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

Πολλές φορές οι όροι τεχνητή νοημοσύνη, μηχανική μάθηση και βαθιά μάθηση χρησιμοποιούνται λάθος ή δεν υπάρχει αρκετή σαφήνεια ως προς το ποια είναι η σχέση μεταξύ τους.

Όπως φαίνεται στο παρακάτω σχήμα, ο τομέας της τεχνητής νοημοσύνης (Artificial Intelligence ή A.I) περιλαμβάνει την μηχανική μάθηση (Machine Learning ή ML) και η βαθιά μάθηση (Deep Learning) είναι υπό-πεδίο της μηχανικής μάθησης.



Εικόνα 2-1: [Σχέσεις μεταξύ πεδίων βαθιάς μάθησης, μηχανικής μάθησης και τεχνητής νοημοσύνης] [ηλεκτρονική εικόνα] Διαθέσιμη: < https://en.wikipedia.org/wiki/Deep_learning#/media/File:AI-ML-DL.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

2.1 Τεχνητή Νοημοσύνη

Μερικές από τις ιδέες που περιγράφονται, αργότερα μέσα στο κείμενο, ανήκουν στο χώρο της τεχνητής νοημοσύνης, αλλά προϋπήρχαν του ίδιου του όρου. Το όνομα του πεδίου παρουσιάστηκε από τον John McCarthy, το 1956 στα πλαίσια της διάσκεψης του κολλεγίου του Dartmouth(Novet, 2017). Από τη δημιουργία του πεδίου μέχρι και σήμερα, εξερευνάται η ιδέα, του αν θα μπορούσαν να δημιουργηθούν υπολογιστές που θα είναι ικανοί να σκέφτονται. Ένας απλός ορισμός που χρησιμοποιήθηκε για να περιγράψει αυτή την προσπάθεια από τον Chollet (2018, σ.4) είναι: «να αυτοματοποιήσουμε ευφυείς διαδικασίες τις οποίες κατά κόρον, τις εκτελούν άνθρωποι».

Ο τρόπος με τον οποίο προσεγγίζεται, πλέον, η τεχνητή νοημοσύνη έχει αλλάξει αρκετά με την πάροδο των χρόνων. Παλιότερα, πολλοί ειδικοί πίστευαν ότι ο μοναδικός τρόπος για να πετύχουμε τέτοιου είδους συμπεριφορά, θα ήταν μέσω του προγραμματισμού ενός αρκετά μεγάλου αριθμού συγκεκριμένων (ρητών) εντολών ώστε να μπορέσει η μηχανή να λάβει αποφάσεις με βάση αυτές, ανάλογα με τα δεδομένα. Αυτή η προσέγγιση έγινε γνωστή και ως συμβολική τεχνητή νοημοσύνη.

Μια τέτοιου είδους προσέγγιση μπορεί να δουλέψει σε περιπτώσεις που τα προβλήματά είναι καλώς ορισμένα, με συγκεκριμένες καταστάσεις, όπως ο προγραμματισμός εφαρμογών που παίζουν σκάκι, αλλά το να βρεθεί ένα σετ κανόνων για πολύ πιο πολύπλοκα προβλήματα όπως η κατηγοριοποίηση εικόνων είναι εξαιρετικά δύσκολο. Σιγά σιγά, για την αντιμετώπιση τέτοιων προβλημάτων, δημιουργήθηκε μια καινούρια προσέγγιση που πήρε τη θέση της συμβολικής τεχνητής νοημοσύνης, η μηχανική μάθηση.

2.2 Μηχανική Μάθηση

Ο όρος μηχανική μάθηση χρησιμοποιείται σε πολύ μεγάλο βαθμό τα τελευταία χρόνια και πολύ κόσμος υποθέτει ότι είναι ένα σχετικά καινούριο κομμάτι στο δρόμο της τεχνητής νοημοσύνης, αλλά δεν ισχύει κάτι τέτοιο. Ακόμα από το 1959 είχε δημοσιευθεί ένα άρθρο από τον Arthur Samuel με τίτλο “Some studies in Machine Learning Using the Game of Checkers”(Samuel, 1959).

Μέσα στο άρθρο αυτό παρουσίασε τον προσπάθειά του να φτιάξει ένα πρόγραμμα το οποίο θα μπορεί να μάθει να παίζει το παιχνίδι ντάμα (checkers) όλο και καλύτερα, χωρίς όμως να το έχει προγραμματίσει με πλήρη σαφήνεια τον τρόπο με τον οποίο θα πετύχει κάτι τέτοιο. Αυτός είναι ένας από τους πολλούς ορισμούς που έχουν

χρησιμοποιηθεί ανά τα χρόνια καθώς δεν υπάρχει ένας καθολικά αποδεκτός ορισμός της μηχανικής μάθησης.

Ο ίδιος ο Samuel, δεν μπορούσε να παίξει καλά το παιχνίδι, αλλά έγραψε ένα πρόγραμμα που θα παίζει δεκάδες χιλιάδες φορές ενάντια στον εαυτό του και κατά τη διάρκεια των παιχνιδιών, το πρόγραμμα, παρατηρούσε ποιες θέσεις πάνω στο ταμπλό οδηγούσαν πιο συχνά σε νίκες παρά σε ήττες και με αυτόν τον τρόπο «μάθαινε» ποιες κινήσεις να προτιμάει. Από κάποιο σημείο και μετά, το πρόγραμμα του Samuel, έμαθε να παίζει το παιχνίδι καλύτερα από τον ίδιο.

Εικόνα 2-2: [Ντάμα (παιχνίδι)] [ηλεκτρονική εικόνα] Διαθέσιμη: < https://upload.wikimedia.org/wikipedia/commons/3/30/International_draughts.jpg> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Στα πλαίσια αυτού του πειράματος, ο Samuel παρατήρησε ότι δεν χρειάζεται να δίνονται πάντα ξεκάθαρες οδηγίες ή εντολές στις μηχανές για να πετύχουν ένα σκοπό αλλά κάποια αρχικά δεδομένα. Με αυτά τα δεδομένα, σωστούς αλγορίθμους και παραμέτρους, οι μηχανές μπορούν να ξεκινήσουν να διακρίνουν πρότυπα και να υπολογίζουν μελλοντικές τιμές. Αυτό το καταφέρνουν επειδή μπορούν και δημιουργούν μοντέλα με βάση τα αρχικά δεδομένα που δώσαμε, τα οποία μπορούμε να

χρησιμοποιήσουμε αργότερα πάνω σε καινούρια δεδομένα ώστε να κάνουμε «προβλέψεις». Η ποιότητα των προβλέψεων αυτών, θα είναι καλύτερη (θα έχει μεγαλύτερη ακρίβεια), όσο καλύτερο είναι το μοντέλο το οποίο αρχικά παράχθηκε.

Εικόνα 2-3: Κλασσικός Προγραμματισμός vs Μηχανική Μάθηση

Ο Tom Mitchell δημιούργησε έναν πιο πρόσφατο ορισμό για τα μηχανική μάθηση, ο οποίος είναι ο εξής: «Ένα πρόγραμμα μαθαίνει από μία εμπειρία E, σχετικά με μια δουλειά T και μια μέτρηση απόδοσης P, αν η απόδοση στο T, όπως μετριέται από το P, βελτιώνει την εμπειρία E» (Mitchell, 1997).

Έχοντας ως παράδειγμα το πρόγραμμα του Samuel, η δουλειά (T) είναι το παίξιμο του παιχνιδιού ντάμα, η μέτρηση της απόδοσης (P) είναι η πιθανότητα να κερδίσει το επόμενο παιχνίδι ενάντια σε κάποιον καινούριο αντίπαλο.

Το πεδίο της μηχανικής μάθησης έχει γίνει ένα από τα πιο δημοφιλή υπο-πεδία της τεχνητής νοημοσύνης, κατά ένα μέρος λόγω της διαθεσιμότητας ολοένα και ταχύτερου υλικού (hardware) που είναι διαθέσιμο στο ευρύ αγοραστικό κοινό, του μεγαλύτερου όγκου διαθέσιμων δεδομένων που μπορεί ο καθένας να βρει διαθέσιμα στο internet αλλά και λόγω άλλων «παραμέτρων» στις οποίες θα αναφερθούμε και αργότερα.

2.3 Νευρωνικά Δίκτυα στη Μηχανική Μάθηση

Η χρήση νευρωνικών δικτύων είναι αρκετά παλιά σαν ιδέα. Ξεκίνησε από την επιθυμία μας να μιμηθούμε τον τρόπο με τον οποίο λειτουργεί ο ανθρώπινος εγκέφαλος. Υπήρχαν περίοδοι που γινόταν μεγάλη χρήση, όπως όταν είχαν πρωτοεμφανιστεί ως ιδέα, αλλά έπειτα για ένα αρκετά μεγάλο χρονικό διάστημα είχαν πέσει στην αφάνεια. Πλέον είναι η πιο σύγχρονη τεχνική που χρησιμοποιείται για την επίλυση διαφόρων προβλημάτων μηχανικής μάθησης. Όπως αναφέρθηκε και πιο πάνω, σε αυτό βοήθησε

και η αύξηση των δυνατοτήτων των σύγχρονων υπολογιστών που μπορούν πλέον να διαχειριστούν την εκπαίδευση νευρωνικών δικτύων μεγάλης κλίμακας.

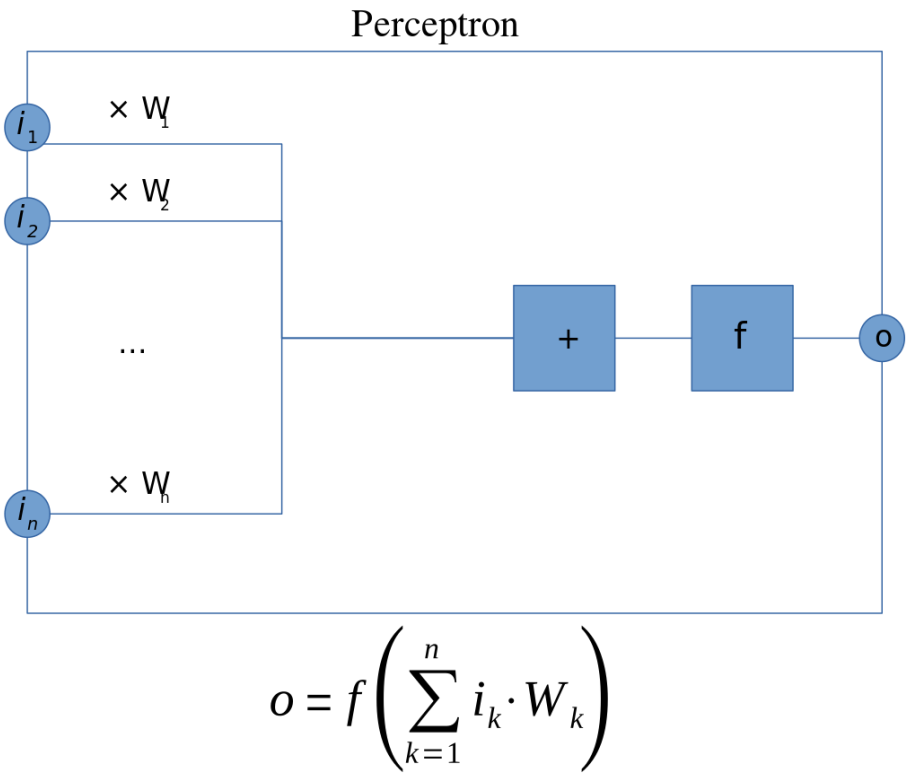
Ο λόγος για τον οποίο χρησιμοποιούνται νευρωνικά δίκτυα είναι για να μπορέσουν να «μάθουν» οι υπολογιστές πιο πολύπλοκες μη-γραμμικές υποθέσεις, από ότι είναι δυνατό κάνοντας χρήση μόνο λογιστικής ή γραμμικής παλινδρόμηση (linear ή logistic regression).

Μέσω της λογιστικής παλινδρόμησης μπορούν να φτιαχτούν υποθέσεις, που είναι αρκετά πολύπλοκες, υψώνοντας κάποια χαρακτηριστικά σε δυνάμεις (x^2, x^4 κλπ) ή συνδυάζοντας χαρακτηριστικά (πολλαπλασιάζοντάς τα), αλλά για μεγάλο αριθμό χαρακτηριστικών κάτι τέτοιο δεν είναι εφικτό καθώς οι υποθέσεις θα κατέληγαν με ένα εξαιρετικά μεγάλο αριθμό καινούριων χαρακτηριστικών που θα οδηγούσε στο πρόβλημα της υπερμοντελοποίησης (overfitting), που θα γίνει περιγραφή και πιο μετά, ή / και σε εξαιρετικά μεγάλη αύξηση του χρόνου που θα έπαιρνε η εκπαίδευση ενός τέτοιου δικτύου. Ιδιαίτερα για προβλήματα κατηγοριοποίησης εικόνων, όπως θα γίνει περιγραφή σε αυτή τη διπλωματική, ο αριθμός των χαρακτηριστικών X που αντιστοιχούν σε μία εικόνα είναι εξαιρετικά μεγάλος.

2.4 Ιστορία των νευρωνικών δικτύων

Το πρώτο μοντέλο νευρωνικού δικτύου δημιουργήθηκε από τον McCulloch και Pits(McCulloch and Pitts, 1943) το 1943 και έκανε ταξινόμηση μεταξύ 2 διαφορετικών κλάσεων, βασιζόμενο σε κάποια είσοδο. Το πρόβλημα με αυτό το αρχικό μοντέλο, ήταν ότι τα βάρη κάθε νευρώνα θα έπρεπε να «ρυθμιστούν» χειρωνακτικά, από έναν άνθρωπο. Κάτι τέτοιο δεν είναι εφικτό, ιδιαίτερα για τα μοντέλα πολλαπλών εκατομμυρίων βαρών, που χρησιμοποιούνται πλέον.

Μετά από μερικά χρόνια, μέσα στη δεκαετία του 1950, δημοσιεύθηκε ο αλγόριθμος Perceptron (Αντίληπτρο) από τον Rosenblatt(Rosenblatt, 1958). Ένα μοντέλο που μπορούσε να μάθει από μόνο του τα βάρη που χρειάζονται για την κατηγοριοποίηση ενός αντικειμένου που έχει δοθεί ως είσοδος, εξαλείφοντας έτσι την ανάγκη της ανθρώπινης παρέμβασης. Η δουλειά που έκανε ο Rosenblatt, δημιούργησε τη βάση για τον στοχαστικό αλγόριθμο φθίνοντος βαθμωτού διανύσματος (stochastic gradient descent algorithm).



Εικόνα 2-4: [Perceptron με εισόδους P, βάρη W, υπολογίζεται το σταθμισμένο άθροισμα πάνω στο οποίο εκτελείται μία συνάρτηση] [ηλεκτρονική εικόνα] Διαθέσιμη:<https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Perceptron.svg/1280px-Perceptron.svg.png > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Έπειτα από αυτή τη δημοσίευση η χρήση τέτοιου είδους τεχνικών είχαν γίνει αρκετά διαδεδομένες αλλά λόγω δημοσιεύσεων που αποδείκνυαν ότι τα Perceptron μπορούν να καταφέρουν μόνο γραμμικές κατηγοριοποιήσεις(Minsky and Papert, 1969), ενώ άλλα ισχυρίζονταν ότι δεν είχαν την απαιτούμενη επεξεργαστική ισχύ για τη δημιουργία μεγάλων νευρωνικών δικτύων, τα νευρωνικά δίκτυα σταμάτησαν να χρησιμοποιούνται για σχεδόν μια δεκαετία.

Εκείνη την περίοδο ξεκίνησαν να χρησιμοποιούνται οι μηχανές διανυσμάτων υποστήριξης (support vector machines), οι οποίες δεν θα περιγραφούν στα πλαίσια αυτής της διπλωματικής εργασίας.

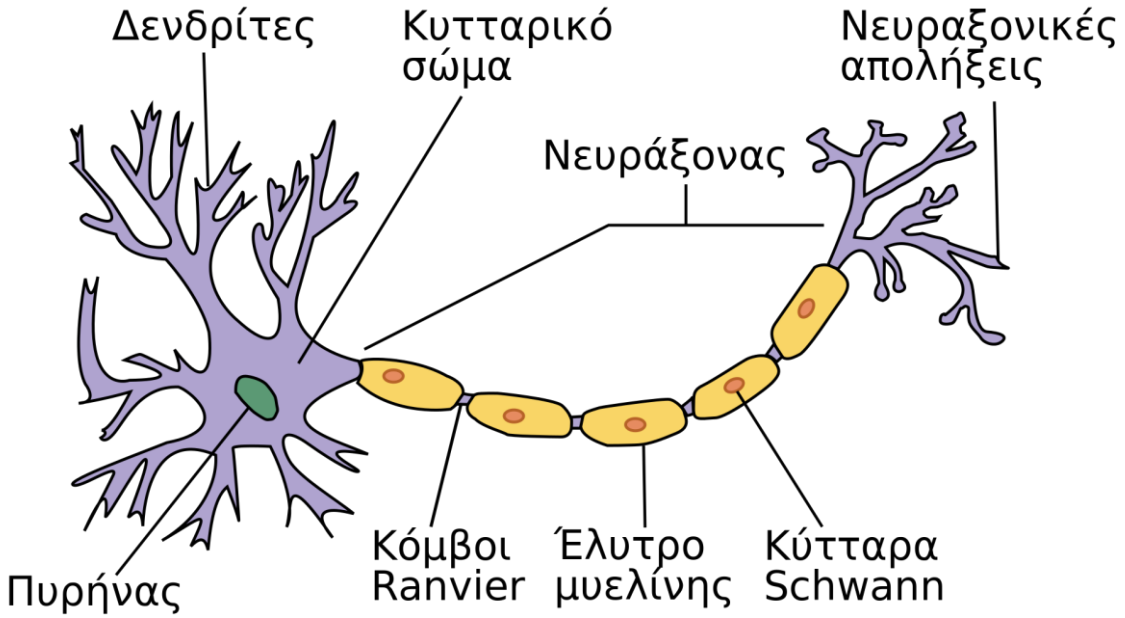
Αυτό που κατάφερε να σώσει και να επαναφέρει τα νευρωνικά δίκτυα στο προσκήνιο της μηχανικής μάθησης, ήταν ο αλγόριθμος «διάδοσης προς τα πίσω» (backpropagation algorithm) από τους Werbos(Werbos, 1975), Rumelhart (Rumelhart et al., 1988)και LeCun(LeCun et al., 1998). Η έρευνα γύρω από αυτό τον αλγόριθμο

αύξησε την αποδοτικότητά του και επέτρεψε την εκπαίδευση νευρωνικών δικτύων, πολλαπλών επιπέδων.

Τα πολλαπλά επίπεδα σε συνδυασμό με τις μη-γραμμικές συναρτήσεις ενεργοποίησης μας επέτρεψαν να λύσουμε μη-γραμμικά προβλήματα. Ο αλγόριθμος backpropagation είναι ο ακρογωνιαίος λίθος των μοντέρνων νευρωνικών δικτύων μέσω του οποίου τα νευρωνικά δίκτυα μπορούν πλέον να μαθαίνουν από τα λάθη τους.

2.5 Αναπαράσταση και Λειτουργία Νευρωνικών Δικτύων

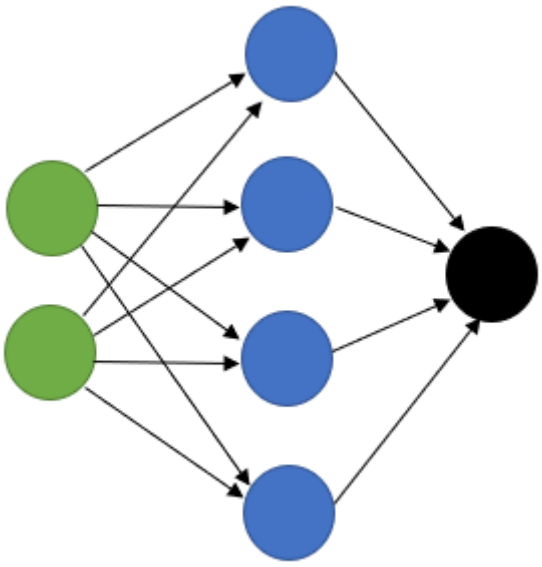
Για να γίνει κατανοητός, ο τρόπος με τον οποίο αναπαριστούνται τα νευρωνικά δίκτυα όσον αφορά την υπόθεση και το μοντέλο μας πρέπει να γίνει αναφορά στον τρόπο λειτουργίας των νευρώνων μέσα στους εγκεφάλους των ανθρώπων, καθώς, θεωρητικά, τα νευρωνικά δίκτυα τους. Πρέπει να σημειωθεί ότι, τα νευρωνικά δίκτυα δεν είναι μια ρεαλιστική αναπαράσταση του εγκεφάλου, αλλά μια μορφή έμπνευσης ώστε να είναι πιο εύκολη η δημιουργία παραλληλισμών μεταξύ ενός πολύ βασικού μοντέλου εγκεφάλου και του τρόπου που γίνεται μίμηση αυτής της συμπεριφοράς μέσω των νευρωνικών δικτύων.



Εικόνα 2-5: [Απεικόνιση νευρώνα του ανθρώπινου εγκεφάλου] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/thumb/0/01/Neuron_el.svg/1920px-Neuron_el.svg.png> [Πρόσβαση στις 31 Αυγούστου 2019]

Κάθε ανθρώπινος εγκέφαλος έχει μερικά δισεκατομμύρια(Herculano-Houzel, 2009) από αυτούς τους νευρώνες. Ο κάθε νευρώνας έχει πολλαπλές «εισόδους» για τα «δεδομένα», τους δενδρίτες, αλλά και ένα (κυτταρικό) σώμα όπου γίνονται οι «πράξεις», ενώ ο νευράξονας λειτουργεί ως έξοδος που στέλνει το αποτέλεσμα των «πράξεων» σε επόμενους νευρώνες μέσω των απολήξεων.

Τα νευρωνικά δίκτυα ακολουθούν την ίδια λογική, έχοντας στην αριστερή πλευρά τα δεδομένα εισόδου (x1,x2,…xn). Αυτό ονομάζεται και ως επίπεδο εισόδου ή επίπεδο 1 (Layer 1). Στο επόμενο επίπεδο, βρίσκονται οι νευρώνες και ονομάζεται το «κρυμμένο» επίπεδο (hidden layer). Μπορούμε να έχουμε παραπάνω από ένα κρυμμένα επίπεδα, για την επίτευξη όλο και πιο περίπλοκων υποθέσεων, και ονομάζονται έτσι επειδή οι τιμές τους είναι «κρυμμένες». Το τελευταίο επίπεδο ονομάζεται το επίπεδο εξόδου (output layer) και είναι το αποτέλεσμα της υπόθεσης.



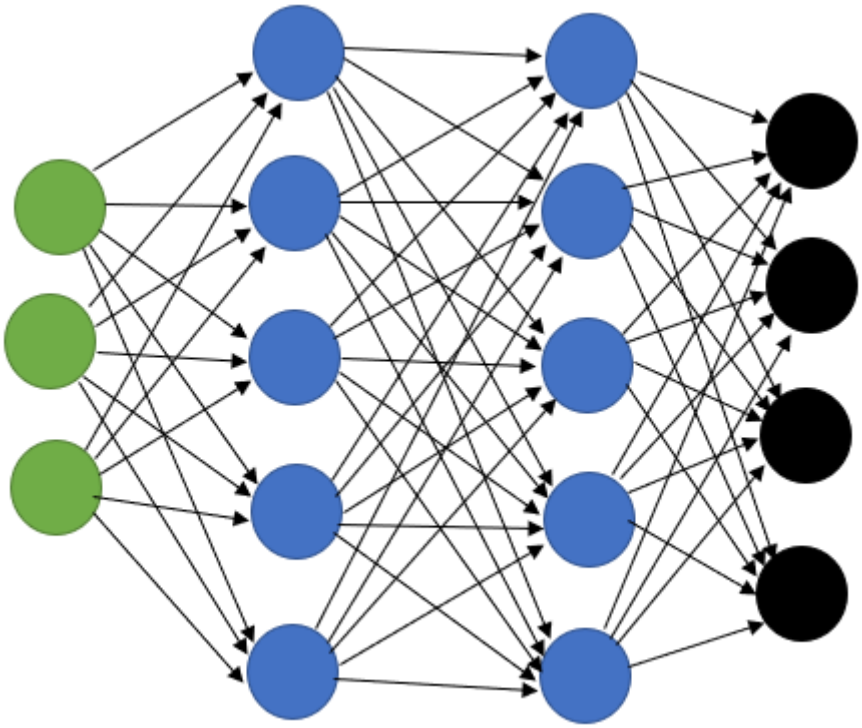
Εικόνα 2-6: Απλό Νευρωνικό Δίκτυο με 2 χαρακτηριστικά εισόδου, 4 νευρώνες στο κρυφό επίπεδο και μία έξοδο

Στο επίπεδο εισόδου οι νευρώνες παίρνουν τις τιμές των αρχικών δεδομένων, ακριβώς όπως αυτές δίνονται. Όπως φαίνεται στην παραπάνω εικόνα, κάθε νευρώνας του πρώτου επιπέδου «συνδέεται», με κάθε νευρώνα του κρυμμένου επιπέδου. Τέτοια δίκτυα λέγονται πλήρως συνδεδεμένα. Κάθε σύνδεση έχει το δικό της βάρος, που μπορεί να είναι είτε θετικό, είτε αρνητικό. Όσο πιο μεγάλη (θετική) τιμή έχει ένα βάρος, τόσο

μεγαλύτερη επιρροή έχει ο ένας νευρώνας (από το προηγούμενο επίπεδο) στον άλλο (στο επόμενο επίπεδο).

Εφόσον κάποιος νευρώνας του «κρυμμένου» επιπέδου, πάρει το αποτέλεσμα όλων των νευρώνων με τους οποίους συνδέεται από το πρώτο επίπεδο, χρησιμοποιεί την εσωτερική του συνάρτηση ενεργοποίησης και αν η τιμή που παραχθεί είναι μεγαλύτερη από κάποιο επίπεδο (συνήθως το 0), τότε μπορεί και αυτό με τη σειρά του να «στείλει» την τιμή του στους νευρώνες του επόμενου επιπέδου με τους οποίους είναι συνδεδεμένο.

Αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου κάθε εσωτερικός νευρώνας στείλει το αποτέλεσμά του στο επίπεδο εξόδου, όπου υπολογίζεται η τελική τιμή του δικτύου. Το δίκτυο της προηγούμενης εικόνας μπορεί να πετύχει κατηγοριοποίηση εικόνων σε δύο κλάσεις, επειδή έχει μόνο ένα νευρώνα εξόδου. Αν η τιμή του νευρώνα αυτού είναι 0 τότε το δίκτυο «πιστεύει» ότι αυτή η εικόνα θα ανήκει στη μία κλάση, ενώ αν το αποτέλεσμα είναι 1, τότε η εικόνα θα ανήκει στην άλλη.



Εικόνα 2-7: Πλήρως συνδεδεμένο δίκτυο με πολλαπλά κρυφά επίπεδα για κατηγοριοποίηση σε 4 κλάσεις

Σε περιπτώσεις κατηγοριοποίησης δεδομένων σε περισσότερες κλάσεις το δίκτυο θα θύμιζε περισσότερο αυτό της προηγούμενης εικόνας με πολλαπλούς νευρώνες εξόδους, όσους και οι επιθυμητές κατηγορίες (στην προκειμένη περίπτωση, 4

κατηγορίες). Οι συναρτήσεις ενεργοποίησης είναι συνήθως διαφορετικές για τους νευρώνες του τελευταίου επιπέδου, ώστε να «παράγουν» μόνο τιμές μεταξύ του 0 και 1 ανάλογα με την κατηγορία που πιστεύει το δίκτυο ότι ανήκει η εκάστοτε εικόνα. Εδώ αν μία εικόνα ανήκει στην $3^{η}$ κατηγορία η έξοδος θα ήταν η εξής: [0 0 1 0].

Μέσω του αλγορίθμου backpropagation διορθώνονται τα αποτελέσματα του νευρωνικού δικτύου κατά τη διάρκεια της εκπαίδευσής του. Εφόσον παραχθεί το τελικό αποτέλεσμα (πρόβλεψη) από το δίκτυο για κάποιο δεδομένο, το αποτέλεσμα συγκρίνεται με αυτό που έπρεπε να είχε δημιουργηθεί.

Σε κάθε περίπτωση, είτε το αποτέλεσμά είναι σωστό, είτε όχι, η πιθανότητα που θα παραχθεί στο τελευταίο επίπεδο πρέπει να είναι η μεγαλύτερη δυνατή για τη «σωστή» κατηγορία. Το ιδεατό σενάριο είναι, σε περίπτωση που γίνεται κατηγοριοποίηση εικόνες αριθμών θα ήταν: αν μια εικόνα απεικονίζει τον αριθμό 2, να γυρνάει το εξής αποτέλεσμα: [0 0 1 0 0 0 0 0 0 0], δηλαδή με πιθανότητα 1 να ανήκει στην κατηγορία 2.

Αυτό που θα κάνει ο αλγόριθμος είναι να προσπαθήσει «πιέσει» τα αποτελέσματα πέραν της κατηγορίας 2 προς τα «κάτω», ενώ παράλληλα να αυξήσει την πιθανότητα να πάρουμε την τιμή 2. Εφόσον το κάθε αποτέλεσμα στο τελευταίο στάδιο είναι συνάρτηση μεταξύ των τιμών των νευρώνων και των βαρών όλων των νευρώνων που συνδέονται με αυτόν, αν η πρόβλεψη ήταν σωστή τότε αυξάνονται οι τιμές των βαρών που «βοήθησαν» ενώ στην αντίθετη περίπτωση μειώνονται τα βάρη τους. έτσι ώστε να φανεί ότι εκείνα τα χαρακτηριστικά δεν οδηγούν στο σωστό αποτέλεσμα.

2.6 Βαθιά Μάθηση

Η βαθιά μάθηση είναι ένα ειδικό υπο-πεδίο της μηχανικής μάθησης, που δίνει έμφαση στη δημιουργία νευρωνικών δικτύων με πολλαπλά διαδοχικά επίπεδα ώστε να δημιουργηθούν όλο και πιο «αφηρημένα χαρακτηριστικά».

Στα πρώτα επίπεδα ενός νευρωνικού δικτύου, έχοντας πάλι ως παράδειγμα την κατηγοριοποίηση εικόνων, εμφανίζονται τα πιο απλά χαρακτηριστικά όπως κάποιες οριζόντιες ή κάθετες γραμμές. Σε επόμενο στάδιο, θα ξεκινήσουν να εμφανίζονται πιο πολύπλοκα χαρακτηριστικά όπως, για παράδειγμα, γωνίες, ενώ σε ακόμα πιο μεγάλο «βάθος» τα χαρακτηριστικά μπορεί να είναι αντίστοιχα με αυτά που αναγνωρίζει ένας άνθρωπος, όπως τα αυτιά ενός σκύλου.

Δεν υπάρχει κάποιος κοινά αποδεκτός αριθμός από επίπεδα, που όταν τον ξεπεράσουμε, αυτόματα μπορεί να θεωρηθεί ένα δίκτυο ως «βαθύ»(Rosebrock, 2017). Αλλά πλέον έχουν εκπαιδευτεί δίκτυα με παραπάνω από 100 επίπεδα όπως το

ResNet(He et al., 2015), χάρη στην ολοένα και μεγαλύτερη υπολογιστική δύναμη, ακόμα και των προσωπικών υπολογιστών αλλά και των περισσότερων διαθέσιμων δεδομένων για εκπαίδευση, τόσο «βαθιών» δικτύων.

Όσο περισσότερο αυξάνεται το «βάθος» ενός νευρωνικού δικτύου, τόσο περισσότερο αυξάνεται η ακρίβεια της κατηγοριοποίησης, σε αντίθεση με άλλους πιο «παραδοσιακούς» αλγορίθμους της μηχανικής μάθησης όπως η λογιστική παλινδρόμηση (logistic regression) όπου από κάποιο σημείο και μετά η απόδοση τους παραμένει σταθερή ακόμα και αν χρησιμοποιούν περισσότερα δεδομένα εκπαίδευσης(Tang et al., 2018).

Λόγω αυτής της σύνδεσης μεταξύ της απόδοσης της βαθιάς μάθησης και την ύπαρξη πολλών δεδομένων, σε κάθε πρόβλημα τέτοιου είδους ο καλύτερος τρόπος τη μεγιστοποίησης της απόδοσης ενός μοντέλου είναι η χρήση όσο δυνατόν περισσότερων δεδομένων.

2.7 Κατηγορίες Μηχανικής Μάθησης

Εφόσον έχει γίνει περιγραφή κάποιων βασικών στοιχείων της μηχανικής μάθησης, πρέπει να γίνει αναφορά στις διαφορετικές κατηγορίες μάθησης καθώς και την περιγραφή προβλημάτων που μπορούν να λυθούν με την κάθε μία εξ αυτών.

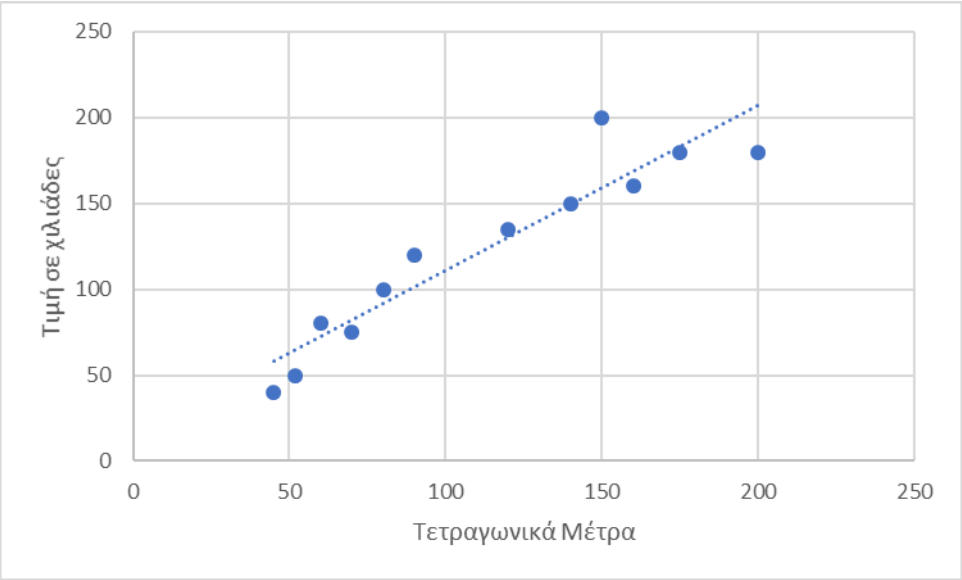
2.7.1 Εποπτευόμενη Μάθηση

Η πρώτη κατηγορία μηχανικής μάθησης και πιθανώς η πιο κοινή, ονομάζεται Εποπτευόμενη Μάθηση (Supervised Learning). Αυτή την κατηγορία χειρίζεται δεδομένα τα οποία ως δεδομένα έχουν έναν αριθμό από διαφορετικά χαρακτηριστικά (τα οποία συνήθως αναπαριστούνται ως X) αλλά και μια κατηγορία όπου ανήκουν (Y).

Η κατηγορία αυτή μπορεί να είναι μια δυαδική αναπαράσταση (π.χ. αν κάποιος ασθενής έχει μία ασθένεια ή όχι) ή ένας αριθμός (π.χ. κατηγοριοποίηση εικόνων που εμπεριέχουν αριθμούς).

Σε τέτοιους είδους προβλήματα, πρέπει να βρεθεί ποια είναι η σχέση μεταξύ των χαρακτηριστικών X και της κατηγορίας Y για τα ήδη υπάρχοντα δεδομένα, ώστε να μπορέσουν να γίνουν προβλέψεις των κατηγοριών που θα ανήκων τα καινούρια δεδομένα για τα οποία θα έχουν μόνο ως πληροφορία τα χαρακτηριστικά τους.

2.7.1.1 Προβλήματα παλινδρόμησης



Εικόνα 2-8: Πρόβλεψη τιμής σπιτιού σε χιλιάδες με μοναδικό χαρακτηριστικό τα τετραγωνικά μέτρα

Το πιο σύνηθες παράδειγμα που χρησιμοποιείται για την καλύτερη κατανόηση της εποπτευόμενης μάθησης, είναι αυτό της πρόβλεψης της τιμής που πρέπει να ζητηθεί για την πώληση ενός ακινήτου. Έχοντας μαζέψει αρκετά δεδομένα για τις τιμές που έχουν πουληθεί ακίνητα σε μία πόλη έχοντας ως μοναδικό κριτήριο τα τετραγωνικά μέτρα και την τιμή που πουληθήκαν τα σπίτια μπορούν να αναπαρασταθούν τα δεδομένα όπως στο προηγούμενο διάγραμμα.

Στη συγκεκριμένη περίπτωση φαίνεται πως λόγω της απλότητας των δεδομένων, μία απλή γραμμική προσέγγιση μπορεί να «αποδώσει» αρκετά καλά αντιστοιχώντας ένα καινούριο σπίτι που είναι 140 τετραγωνικά μέτρα, η τιμή που προβλέπεται είναι γύρω στις 150 χιλιάδες ευρώ.

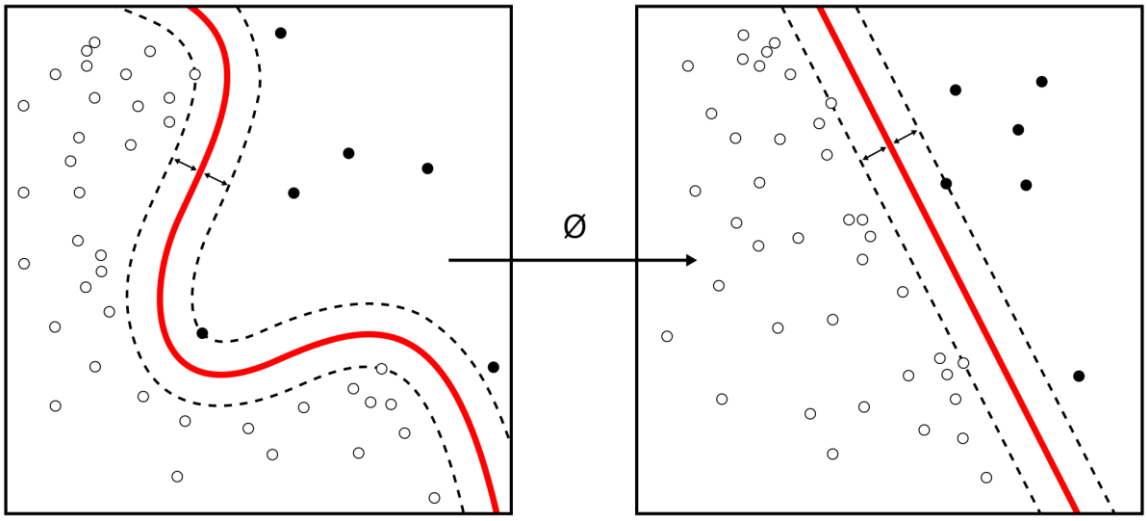
Αυτού του είδους τα προβλήματα ονομάζονται προβλήματα παλινδρόμησης (regression problems) καθώς γίνεται πρόβλεψη μίας συνεχής τιμής (εδώ την τιμή του ακινήτου).

2.7.1.2 Προβλήματα κατηγοριοποίησης

Η δεύτερη από τις κατηγορίες προβλημάτων της εποπτευόμενης μάθησης, είναι τα προβλήματα κατηγοριοποίησης, όπου η τιμή που πρέπει προβλεφθεί είναι ένας διακριτός αριθμός.

Τέτοιου είδους προβλήματα μπορεί να αφορούν κάποιου είδους ασθένεια, όπου είτε οι ασθενείς είναι φορείς της ή όχι., αλλά υπάρχουν και περιπτώσεις που μπορεί να

υπάρχουν παραπάνω των δύο κατηγοριών. Σε κάθε περίπτωση όμως οι κατηγορίες πρέπει να παραμένουν διακριτές μεταξύ τους.



Εικόνα 2-9: [Δεδομένα εκπαίδευσης με όριο απόφασης] [ηλεκτρονική εικόνα] Διαθέσιμη:<https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Kernel_Machine.svg/1920px-Kernel_Machine.svg.png > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Αναλόγως με την πολυπλοκότητα του προβλήματος, τόσο πιο περίπλοκος αλγόριθμος πρέπει να χρησιμοποιηθεί για την σωστή κατηγοριοποίηση των δεδομένων.

Όπως φαίνεται στο δεξί κομμάτι της παραπάνω εικόνας, μόνο ένας απλός γραμμικός διαχωρισμός είναι αρκετός για την σωστή εξαγωγή συμπερασμάτων, ενώ στην αριστερή πρέπει να χρησιμοποιηθεί ένα πιο πολύπλοκο μοντέλο.

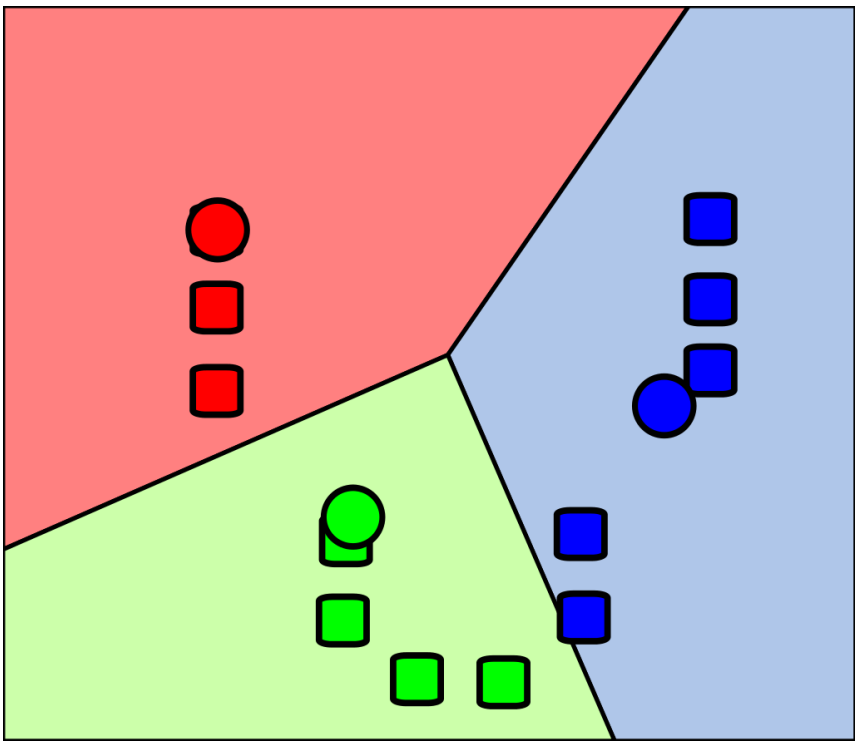
Κάθε πρόβλημα ανάλογα με τα χαρακτηριστικά του χρειάζεται μια διαφορετική προσέγγιση, ενώ χρησιμοποιούνται και διάφορες μετρικές όπως η ακρίβεια για να εξακριβωθεί η ποιότητα της λύσης.

Τέτοιου είδους πρόβλημα θα λυθεί στα πλαίσια της διπλωματικής εργασίας.

2.7.2 Μη-εποπτευόμενη Μάθηση

Η δεύτερη μεγάλη κατηγορία μηχανικής μάθησης, είναι αυτή της μη-εποπτευόμενης μάθησης. Σε αντίθεση με την εποπτευόμενη μάθηση όπου τα δεδομένα είχαν κάποια κλάση, εδώ τα δεδομένα εκπαίδευσης δεν έχουν μία κατηγορία στην οποία ανήκουν.

Σε αυτή την κατηγορία προβλημάτων, ο αλγόριθμος που θα χρησιμοποιηθεί, προσπαθεί να βρει κάποια δομή μέσα στα δεδομένα, δηλαδή να χωρίσει τα δεδομένα σε ομάδες (clusters), ανάλογα με τα κοινά χαρακτηριστικά τους.



Εικόνα 2-10: [Clustering με k-means] [ηλεκτρονική εικόνα] Διαθέσιμη:< https://upload.wikimedia.org/wikipedia/commons/thumb/d/d2/K_Means_Example_Step_4.svg/1024px-K_Means_Example_Step_4.svg.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Μέσω της μη-εποπτευόμενης μάθησης μπορούν να ανακαλυφθούν πρότυπα που δεν ξέραμε ότι υπήρχαν μέσα στα δεδομένα. Ένα παράδειγμα εταιρείας που χρησιμοποιεί τέτοιου είδους μηχανικής μάθησης είναι η Google για την ιστοσελίδα της Google News. Η Google βρίσκει χιλιάδες διαφορετικές ειδήσεις σε διαφορετικές ιστοσελίδες και αυτόματα ομαδοποιεί αυτές που είναι σχετικές μεταξύ τους (η ίδια ιστορία από διαφορετικούς ιστοτόπους) και τις παρουσιάζει με συνδέσμους προς όλες τις πηγές.

Μία άλλη εξαιρετική χρήση αυτού του είδους μηχανικής μάθησης είναι στο πρόβλημα του πάρτι με τα κοκτέιλ (Cocktail party problem)(Cherry, 1953). Σε αυτό το πρόβλημα καλούμαστε να ξεχωρίσουμε τις φωνές που ακούγονται μέσα σε ένα πάρτι έχοντας αφήσει μικρόφωνα σε διαφορετικά σημεία του δωματίου.

2.7.3 Ημι-εποπτευόμενη μάθηση

Σε περιπτώσεις όπου η πληροφορία όσον αφορά την κλάση των δεδομένων, υπάρχει για μερικά από αυτά αλλά όχι για, τότε υπάρχει μία άλλη υβριδική μορφή μάθησης, η ημι-εποπτευόμενη μάθηση.

Μια τέτοιου είδους προσέγγιση μπορεί να χρησιμοποιηθεί σε περιπτώσεις που σε ένα πρόβλημα υπάρχει ένας μικρός όγκος δεδομένων με γνωστή κλάση, αλλά υπάρχουν επίσης και πολλά άλλα που για κάποιο λόγο δεν έχουν κατηγοριοποιηθεί (ίσως η διαδικασία να απαιτεί δυσανάλογα πολλές εργατοώρες). Οι αλγόριθμοι σε αυτές τις περιπτώσεις αναλύουν τα κατηγοριοποιημένα δεδομένα και προσπαθούν να κατηγοριοποιήσουν τα υπόλοιπα, ώστε να χρησιμοποιηθούν και αυτά με τη σειρά τους σαν δεδομένα εκπαίδευσης.

Όπως είναι φυσικό τέτοιου είδους προσεγγίσεις, οδηγούν σε χαμηλότερη ακρίβεια από ότι αν όλα τα δεδομένα μας είναι κατηγοριοποιημένα, αλλά απαιτούν λιγότερη «προσπάθεια» από την «χειρωνακτική» εναλλακτική.

2.7.4 Ενισχυμένη Μάθηση

Σε αντίθεση με τις προηγούμενες κατηγορίες, όπου εφόσον είχε παραχθεί ένα, τελικό, μοντέλο αρκετά υψηλής ακρίβειας, χρησιμοποιείται αυτούσιο από εκείνο το σημείο, στα προβλήματα της ενισχυμένης μάθησης (reinforcement learning), το παραγόμενο μοντέλο συνεχίζει να βελτιώνεται χρησιμοποιώντας καινούρια δεδομένα κατά τη διάρκεια χρήσης του, εμφανίζοντας βελτιώσεις σε σχέση με τις . προηγούμενες «εκδόσεις» του.

Στην ενισχυμένη μάθηση το αποτέλεσμα του μοντέλου δεν έχει μια κατηγορία αλλά ένα βαθμό ανάλογα με το αποτέλεσμα που έβγαλε. Ένας αλγόριθμος που χρησιμοποιείται σε τέτοιου είδους προβλήματα είναι ο Q-learning.

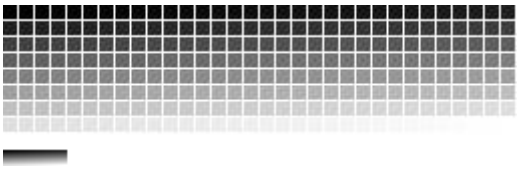
2.8 Εικόνες

Έχοντας ήδη δει ότι τα νευρωνικά δίκτυα χρειάζονται κάποιου είδους δεδομένα ως είσοδο για να δημιουργήσουν τις προβλέψεις τους, και εφόσον θα χρησιμοποιηθούν εικόνες για το πρόβλημα της διπλωματικής, πρέπει να γίνει περιγραφή του τρόπου με τον οποίο αντιλαμβάνονται οι υπολογιστές τις εικόνες και πως τις αναπαριστούν.

2.8.1 Εικονοστοιχεία

Τα εικονοστοιχεία ή pixels είναι τα βασικά στοιχεία που εμπεριέχει μία εικόνα. Κάθε εικόνα μπορεί να θεωρηθεί ως ένα πλέγμα, όπου το κάθε τετράγωνο είναι ένα εικονοστοιχείο. Όταν αναφερόμαστε στην ανάλυση μιας εικόνας ως 1920 επί 1080, αυτό σημαίνει ότι εμπεριέχει 1920 εικονοστοιχεία οριζοντίως και 1080 καθέτως. Η αναπαράσταση μέσα στους υπολογιστές γίνεται με πίνακες που στη συγκεκριμένη περίπτωση θα έχει 1920 στήλες και 1080 γραμμές, άρα συνολικά έχει 1920 * 1080 = 2.073.600 διαφορετικές τιμές.

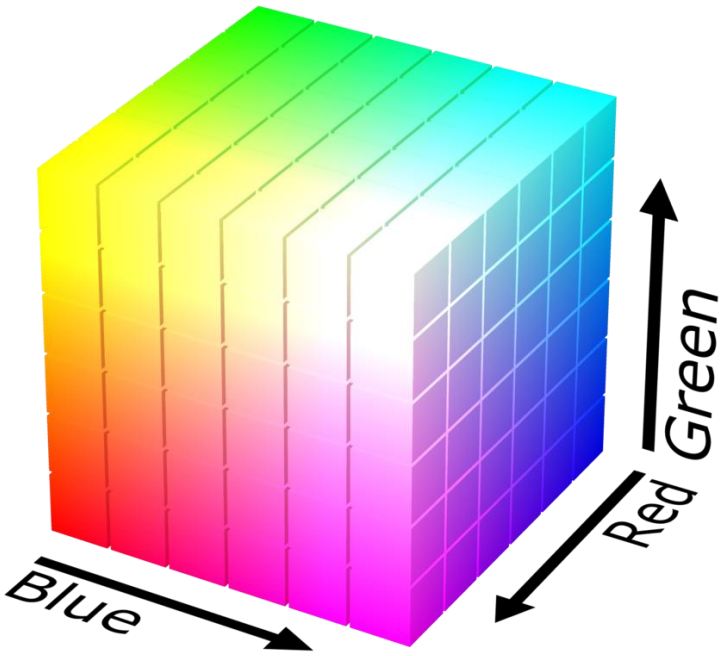
Ανάλογα με αν η εικόνα είναι ασπρόμαυρη ή έγχρωμη, τα εικονοστοιχεία αναπαριστούνται διαφορετικά. Σε μία ασπρόμαυρη φωτογραφία, το κάθε ένα εικονοστοιχείο έχει μία μοναδική τιμή από το 0 έως το 255, όπου το 0 είναι το μαύρο ενώ το 255 το λευκό. Όλες οι ενδιάμεσες τιμές είναι διαφορετικές αποχρώσεις του γκρι που όσο αυξάνεται η τιμή τους, τόσο αυξάνεται και η «φωτεινότητά» τους.



Εικόνα 2-11: [Αναπαράσταση εικονοστοιχείων ασπρόμαυρων εικόνων] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/b/be/Grayscale_8bits_palette.png > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Για τις έγχρωμες εικόνες, χρησιμοποιείται μια λίστα με τρεις διαφορετικές τιμές για το κάθε εικονοστοιχείο, καθώς οι υπολογιστές αναπαριστούν όλα τα χρώματα κάνοντας συνδυασμούς μεταξύ του κόκκινου, πράσινου και μπλε (RGB). Πάλι για το κάθε χρώμα χρησιμοποιούνται τιμές από το 0 έως το 255, όπου το 0 δείχνει ότι το συγκεκριμένο χρώμα δεν χρησιμοποιείται καθόλου ενώ στο 255 χρησιμοποιείται πλήρως.

Όσο πιο υψηλές τιμές έχουν και τα τρία αυτά χρώματα τόσο πιο «φωτεινό» είναι το κάθε εικονοστοιχείο, άρα η χρήση και των τριών με τιμή 255 θα δώσει το λευκό χρώμα.



Εικόνα 2-12: [Αναπαράσταση χρωμάτων στο RGB σε ένα κύβο] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/RGB_color_solid_cube.png/1280px-RGB_color_solid_cube.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Λόγω των 3 διαφορετικών τιμών πολλά εργαλεία στην Python, όπως το numpy αναπαριστούν προγραμματιστικά την κάθε εικόνα ως πολυδιάστατους πίνακες με πλάτος και ύψος ανάλογο με την ανάλυση της εικόνας και βάθος 3.

Καθώς η Python χρησιμοποιεί το 0 για την αρχή των πινάκων της, οι εικόνες ξεκινάνε την αρίθμηση από την πάνω αριστερά γωνία με το πρώτο εικονοστοιχείο να είναι στις «συντεταγμένες» (0,0) και το «τελευταίο» εικονοστοιχείο θα είναι το κάτω δεξιά, με συντεταγμένες για μια εικόνα 1920 x 1080, (1919,1079).

2.8.2 Normalization

Μία τεχνική που χρησιμοποιούμε για να μειώσουμε τον χρόνο εκπαίδευσης των δικτύων, όταν χρησιμοποιούμε εικόνες για δεδομένα εκπαίδευσης είναι να διαιρούμε την τιμή του κάθε εικονοστοιχείου με τη μέγιστη δυνατή τιμή, το 255, ώστε όλες οι τιμές της εικόνας να είναι μεταξύ το 0 και 1.

Με αυτό τον τρόπο μπορούμε να είμαστε σίγουροι ότι οι πράξεις που θα χρειαστεί να κάνει το δίκτυο κατά την εκπαίδευσή του, δεν θα ξεφύγουν σε κλίμακα, και χρόνο υπολογισμού. Αυτή η διαδικασία ονομάζεται κανονικοποιήση (normalization).

2.9 Convolutional Neural Networks

2.9.1 Γιατί δεν χρησιμοποιούμε πλήρες συνδεδεμένα δίκτυα

Οι εικόνες εμπεριέχουν πάρα πολύ πληροφορία μέσα τους. Αυτό δημιουργεί δύο διαφορετικά προβλήματα σε περιπτώσεις που χρησιμοποιούνται πιο παραδοσιακά νευρωνικά δίκτυα.

Το πρώτο πρόβλημα είναι ο τεράστιος αριθμός από βάρη τα οποία θα πρέπει να εκπαιδευτούν σε κάθε επίπεδο. Για μία έγχρωμη εικόνα 200 επί 200 εικονοστοιχείων θα είναι (200 * 200 * 3) 120.000 επί τον αριθμό των νευρώνων στο πρώτο κρυφό επίπεδο. Κάτι τέτοιο «κοστίζει» πολύ υπολογιστικά και τέτοιου είδους δίκτυα θα έκαναν πολλή ώρα να εκπαιδευτούν.

Το δεύτερο πρόβλημα είναι θεωρητικά, επέκταση του πρώτου, όπου λόγω του μεγάλου αριθμού βαρών που υπάρχουν μέσα στο δίκτυο, είναι σχεδόν σίγουρο ότι θα οδηγηθούμε σε καταστάσεις over-fitting.

2.9.2 Γιατί χρησιμοποιούμε Convolutional Neural Networks

Στα convolutional neural networks (συνελεκτικά νευρωνικά δίκτυα) ο κάθε νευρώνας ενός επιπέδου συνδέεται μόνο με μία μικρή περιοχή του προηγούμενου επιπέδου, αντί να συνδέεται με όλους τους προηγούμενους νευρώνες, μειώνοντας έτσι κατά πολύ συνολικό αριθμό βαρών που πρέπει μάθει το δίκτυο.

Τα CNNs είναι ένας τύπος εμπροσθοτροφοδοτούμενου (feed-forward) νευρωνικού δικτύου, όπου σε κάθε επίπεδο χρησιμοποιείται ένας αριθμός από διαφορετικά φίλτρα, συνδυάζονται τα αποτελέσματα και το αποτέλεσμα δίνεται ως είσοδος για το επόμενο επίπεδο. Τα CNNs μπορούν και μαθαίνουν από μόνα τους τις τιμές αυτών των φίλτρων.

Τα φίλτρα αυτά στα πρώτα επίπεδα μπορούν να αντιληφθούν αρχικά απλή πληροφορία, που υπάρχουν κάθετες και που υπάρχουν οριζόντιες γραμμές και όσο προσθέτουμε κρυφά επίπεδα τόσο πιο πολύπλοκη πληροφορία θα μπορούν να διακρίνουν.

Το τελευταίο επίπεδο του CNN θα χρησιμοποιήσει τα πιο πολύπλοκα χαρακτηριστικά που έχει «μάθει» το δίκτυο, για να καταλάβει το περιεχόμενο της εικόνας. Αυτή η διαδικασία, δημιουργίας, αυτομάτως όλο και πιο περίπλοκα χαρακτηριστικά είναι παρόμοια με μαθηματικές συναρτήσεις που χρησιμοποιούν το αποτέλεσμα μιας προηγούμενης (π.χ. f(g(x))) και κάνουν τα CNNs εξαιρετικά χρήσιμα σε τέτοιου είδους προβλήματα.

2.9.3 Convolutions

Τα convolutions είναι ένας όρος που χρησιμοποιείται και σε πεδία εκτός του computer vision, όπως στην επεξεργασία εικόνων, σε δουλείες όπως το «θόλωμα» μιας εικόνας όπως και στην ανίχνευση του περιγράμματος αντικειμένων μέσα σε μια εικόνα.

Τα convolutions στα πλαίσια της μηχανικής μάθησης είναι οι πολλαπλασιασμοί μεταξύ δύο πινάκων, στοιχείο με στοιχείο (element wise multiplication) ακολουθούμενα από ένα άθροισμα. Οι πίνακές θα πρέπει να έχουν τις ίδιες διαστάσεις.

Έχοντας ήδη δει πως αντιλαμβάνονται οι υπολογιστές τις εικόνες μπορούμε να καταλάβουμε ότι ο πρώτος πίνακας θα είναι η ίδια η εικόνα. Ο δεύτερος πίνακας θα είναι το «φίλτρο» που θα περάσει πάνω από όλη την εικόνα, από αριστερά προς δεξιά και από πάνω προς τα κάτω, ώστε να «πάρουμε» τα αποτελέσματα που μας ενδιαφέρουν σε έναν καινούριο πίνακα (για παράδειγμα σε περίπτωση ενός προγράμματος επεξεργασίας εικόνων που θέλει να τη «θολώσει» θα έχει έτοιμο ένα φίλτρο με έτοιμες τιμές που θα κάνουν αυτή τη διαδικασία).

13	168	232	166	183	193
250	87	246	31	129	150
202	125	165	131	16	232
91	122	224	111	215	181
245	226	73	178	184	113
157	194	24	169	212	77

X

1	0	-1
1	0	-1
1	0	-1

=

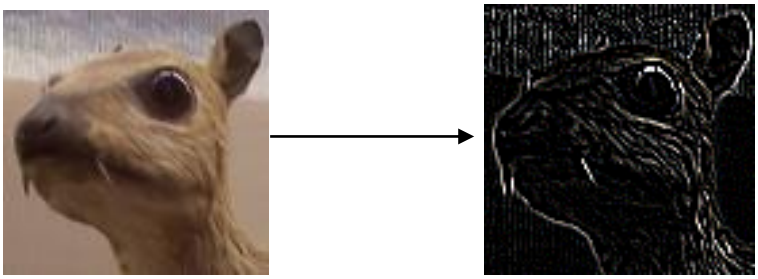
-178	52	315	-247
-92	61	275	-290
76	53	47	-106
172	84	-290	87

Εικόνα 2-13: Παράδειγμα ενός convolution

Στην πιο πάνω εικόνα φαίνεται, ένα από τα πιο συνηθισμένα παραδείγματα χρήσης ενός απλό φίλτρου για την ανίχνευση κατακόρυφων ακμών σε μια εικόνα. Ο υπολογισμός του πρώτου αριθμού πάει έτσι όπως παρουσιάστηκε και πιο πάνω. Πολλαπλασιάζουμε κάθε αριθμό του φίλτρου με τον αριθμό του εικονοστοιχείου που «ακουμπάει» όσο το περνάμε πάνω από όλη την εικόνα. Εδώ έχουμε: 13*1 + 250*1 + 202*1 + 168*0 + 87*0 + 125*0 + 232*(-1) + 246*(-1) + 165*(-1) = 13 + 168 + 87 -232 - 246 – 165 = -178.

Αντίστοιχα παίρνουμε και υπόλοιπες τις τιμές βήμα βήμα, για όλο τον υπόλοιπο καινούριο πίνακα. Στις έγχρωμες εικόνες αυτή η διαδικασία θα χρειαστεί να γίνει 3 φορές, μία για κάθε χρώμα (RGB).

Μπορεί να μην είναι προφανές στους αριθμούς της εικόνας, αλλά σε περιπτώσεις όπου έχουμε απότομη μεταβολή των αριθμών από τις αριστερές στήλες, στις δεξιές (σε περιπτώσεις δηλαδή που για παράδειγμα ξεκινάει κάποιο άλλο αντικείμενο στην εικόνα) αυτό το φίλτρο θα «σκουράνει» την εικόνα αριστερά και δεξιά της μεταβολής, ενώ θα αυξήσει την φωτεινότητα στο σημείο της μετάβασης.



Εικόνα 2-14: [Εφαρμογή φίλτρου για κάθετες και οριζόντιες γραμμές] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/5/50/Vd-Orig.png & https://upload.wikimedia.org/wikipedia/commons/6/6d/Vd-Edge3.png > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

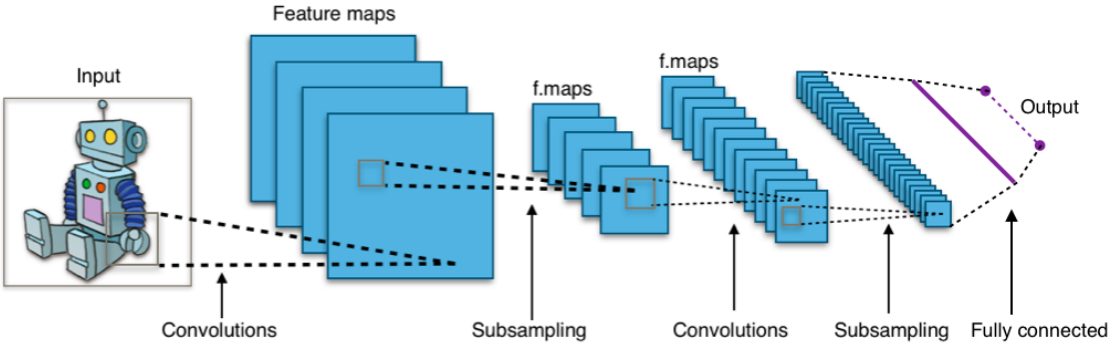
Τα φίλτρα που χρησιμοποιούνται στα CNNs είναι συνήθως τετράγωνοι πίνακες ώστε να κάνουν χρήση των βελτιστοποιημένων βιβλιοθηκών γραμμικής άλγεβρας που δουλεύουν καλύτερα σε τετράγωνους πίνακες, οι πιο δημοφιλείς διαστάσεις είναι 3 επί 3, 5 επί 5 και 7 επί 7 (συνήθως στο πρώτο επίπεδο του δικτύου για να μειωθούν γρήγορα τις διαστάσεις των εικόνων χωρίς τη χρήση του max pooling αλλά αυτή είναι μια πιο «προηγμένη» τεχνική).

Όπως φαίνεται και από τους πίνακες οι διαστάσεις των εικόνων πέφτουν κάθε φορά που χρησιμοποιούνται τέτοια φίλτρα, αλλά υπάρχουν τρόποι για να αντιμετωπιστεί αυτό το πρόβλημα σε περίπτωση που τα δεδομένα περιλαμβάνουν μόνο εικόνες μικρών διαστάσεων, όπως το padding, που περιγράφεται παρακάτω.

2.9.4 Επίπεδα των CNNs

Τυπικά τα CNNs έχουν τις παρακάτω κατηγορίες επιπέδων:

1) Convolution
2) Activation
3) Pooling
4) Fully Connected
5) Batch Normalization
6) Dropout



Εικόνα 2-15: [Αρχιτεκτονική ενός CNN] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

2.9.4.1 Convolutional Layer

Αυτού του είδους τα επίπεδα είναι τα κύρια σε κάθε CNN, κάθε επίπεδο έχει ένα αριθμό από φίλτρα (τα convolutions που περιεγραφήκανε παραπάνω), και μόνο στο πρώτο πρέπει να δηλωθούν και οι διαστάσεις των εικόνων που θα χρησιμοποιηθούν, ως είσοδος. Όλες οι εικόνες θα πρέπει να έχουν προ-επεξεργαστεί σε περίπτωση που δεν έχουν τις ίδιες διαστάσεις.

Τα φίλτρα θα φτιάξουν ίσο αριθμό από δυσδιάστατους πίνακες, κάνοντας το καθένα τις πράξεις που είδαμε προηγουμένως. Οι πίνακες αυτοί ονομάζονται activation maps (χάρτες ενεργοποίησης). Εφόσον ολοκληρώσουν όλα τα φίλτρα, θα φτιαχτούν τα δεδομένα εισόδου του επόμενου πεδίου, που θα είναι ένας πίνακας με βάθος όσο και ο αριθμός τους.

Κάθε πίνακας είναι το αποτέλεσμα ενός νευρώνα που «κοιτάει» σε ένα μικρό κομμάτι της εικόνας, έτσι το δίκτυο «μαθαίνει» φίλτρα που ενεργοποιούνται όταν «βλέπουν» ένα συγκεκριμένο χαρακτηριστικό σε συγκεκριμένα κομμάτια της εικόνας.

Επίσης μπορούν να δηλωθούν και το πόσο μεγάλα «βήματα» θα κάνουν τα φίλτρα, όσο «προχωράνε» μέσα στην εικόνα. Τα «βήματα» αυτά (strides) είναι συνήθως 1 (το φίλτρο προχωράει κατά ένα στοιχείο προς τα δεξιά, όπως στο παράδειγμα

παραπάνω) ή 2. Με μικρά βήματα συγκρατείται το μεγαλύτερο μέρος της πληροφορίας αλλά και μεγαλύτερους πίνακες εξόδου, σε αντίθεση με τα μεγαλύτερα «βήματα» που οι πίνακες μπορεί να μικραίνουν πολύ πιο γρήγορα. Με αυτό τον τρόπο χάνουμε κάποια από την πληροφορία της εικόνας αλλά να βοηθήσει στη μείωση του χρόνου εκπαίδευσης, σε περιπτώσεις όπου τα δεδομένα περιλαμβάνουν εικόνες αρκετά υψηλής ανάλυσης.

Σε περιπτώσεις όπου η ανάλυση των εικόνων πρέπει να παραμείνει σταθερή, είτε επειδή οι εικόνες είναι μικρές, ή επειδή χρησιμοποιούνται πολλαπλά convolutional επίπεδα, μπορεί να γίνει χρήση κάποιου είδους «γεμίσματος» (padding). Με το padding, προστίθενται μία σειρά παραπάνω στοιχείων στον τελικό πίνακα γύρω από την εικόνα. Φίλτρα 3 επί 3, μειώνουν κατά δύο την ανάλυση, όπως στο παράδειγμα, από ανάλυση 6 επί 6 σε 4 επί 4. Άρα με padding 1 θα δημιουργείται πάλι ένας πίνακας 6 επί 6, βάζοντας γύρω από τον πίνακα τις καινούριες τιμές.

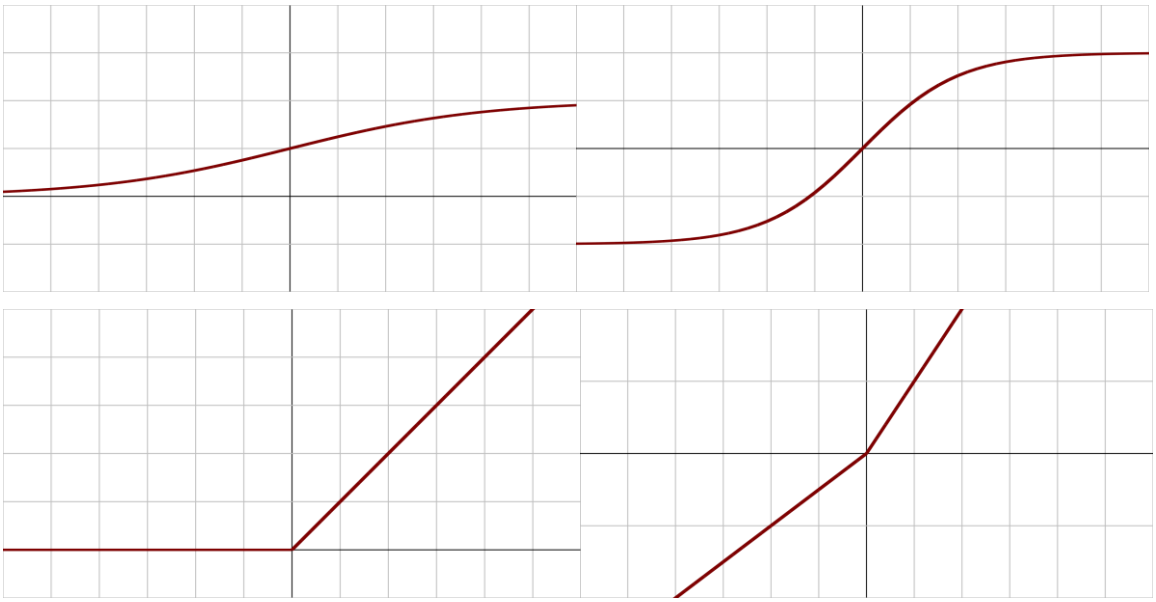
Μία δημοφιλής μορφή padding είναι το zero padding που απλά γίνεται εισαγωγή της τιμής 0 γύρω από την τελική εικόνα. Με αυτό τον τρόπο διατηρούνται οι διαστάσεις της εικόνας αλλά δεν χρειάζεται παραπάνω χρόνος για τον υπολογισμό πιο πολύπλοκων τιμών, σε διαφορετικού είδους padding, αλλά χάνεται κάποιος μέρος της πληροφορίας.

0	0	0	0	0	0	0	0
0	237	218	54	196	58	126	0
0	101	116	22	76	9	158	0
0	253	52	138	213	169	251	0
0	117	136	70	162	199	70	0
0	233	36	113	70	10	168	0
0	191	248	144	12	138	235	0
0	0	0	0	0	0	0	0

Εικόνα 2-16: Zero padding

2.9.4.2 Activation Layer

Μετά από κάθε convolutional επίπεδο, υπάρχουν τα επίπεδα ενεργοποίησης, όπου χρησιμοποιούνται συναρτήσεις όπως η ReLU, ELU, tanh και άλλες. Αυτό το επίπεδο δεν έχει παραμέτρους που χρειάζεται να μάθει το δίκτυο.



Εικόνα 2-17: [Διαγράμματα συναρτήσεων ενεργοποίησης, από πάνω αριστερά προς τα κάτω δεξιά: σιγμοειδής, tanh, ReLu, LeakyReLu] [ηλεκτρονική εικόνα] Διαθέσιμες: <https:// https://en.wikipedia.org/wiki/Activation_function > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Οι συναρτήσεις ενεργοποίησης, παίρνουν το αποτέλεσμα του convolutional επιπέδου, τον πίνακα, και αλλάζουν τις τιμές.

Η συνάρτηση ενεργοποίησης που χρησιμοποιείται πιο συχνά στα κρυφά επίπεδα, είναι η ReLU που μηδενίζει όλες τις αρνητικές τιμές του πίνακα, ενώ κρατάει τις θετικές ακριβώς όπως τις πήρε (γραμμικό θετικό κομμάτι).

Κάθε συνάρτηση έχει τα αρνητικά και θετικά της, οι πιο παραδοσιακές συναρτήσεις όπως η σιγμοειδής και tanh, μπορεί να μειώσουν πάρα πολύ την «κλίση» (gradient) του προβλήματός μας καθώς και οι συναρτήσεις που είναι κεντραρισμένες γύρω από το 0, όπως η tanh, προτιμώνται επειδή κάνουν την εκπαίδευση στο επόμενο επίπεδο πιο εύκολη.

Όλες αυτές οι μη γραμμικές συναρτήσεις χρησιμοποιούνται ώστε να καταφέρουμε να πάρουμε πιο πολύπλοκα μοντέλα από ότι αν χρησιμοποιούσαμε γραμμικές συναρτήσεις που το μοντέλο μας θα ήταν μια απλή λογιστική παλινδρόμηση.

Σε αντίθεση με τα κρυφά επίπεδα, οι συναρτήσεις ενεργοποίησης που χρησιμοποιούνται στο τελευταίο επίπεδο είναι συνήθως είτε η σιγμοειδής, για

προβλήματα που μία δυαδική αναπαράσταση είναι αρκετή, καθώς το αποτέλεσμα πάει από 0 έως 1, είτε χρησιμοποιείται η softmax.

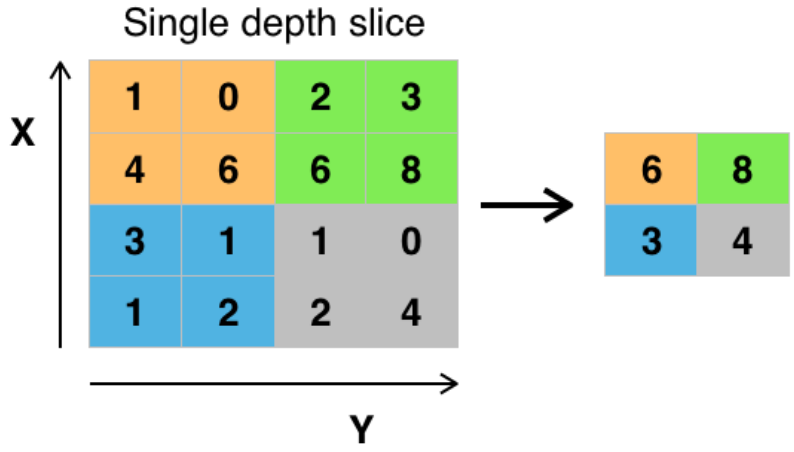
Η softmax επιστρέφει την πιθανότητα μίας εικόνας να ανήκει σε μία κατηγορία, όλες οι τιμές που θα γυρίσει θα έχουν άθροισμα 1. Η πρόβλεψη του μοντέλου, είναι η επιλογή της μεγαλύτερης πιθανότητας για κάθε εικόνα. Αν για παράδειγμα επιστρέψει τις τιμές 0.7 για μία εικόνα ότι είναι γάτα, 0.2 ότι είναι σκύλος και 0.1 ότι είναι άλογο, η εικόνα θα ταξινομηθεί σαν γάτα.

2.9.4.3 Pooling Layer

Ο αριθμός των χαρακτηριστικών σε δίκτυα που εκπαιδεύονται με εικόνες είναι εξαιρετικά μεγάλος, επειδή οι εικόνες περιλαμβάνουν πάρα πολλή πληροφορία.

Σε αυτό το επίπεδο γίνεται προσπάθεια σμίκρυνσης της εικόνας χωρίς όμως να χαθεί χρήσιμη πληροφορία. Για να επιτευχθεί αυτό το αποτέλεσμα, σε αυτό το επίπεδο δημιουργείται ένας μικρός πίνακας (συνήθως 2 επί 2, ή 3 επί 3) και με «βήμα» (συνήθως 1 ή 2). Έπειτα ο πίνακας μετακινείται πάνω στην εικόνα, όπως τα φίλτρα του convolutional επιπέδου, από πάνω αριστερά μέχρι και κάτω δεξιά (από αριστερά προς τα δεξιά, μέχρι να τελειώσει μία γραμμή και μεταφέρεται στην από κάτω γραμμή, στην αριστερή πλευρά του πίνακα).

Πάνω στο κάθε σημείο της εικόνας που «κάθεται», ψάχνει και «κρατάει» μόνο τη μέγιστή τιμή του. Αυτή η διαδικασία γίνεται για όλα τα φίλτρα που δημιουργήθηκαν στο προηγούμενο επίπεδο.



Εικόνα 2-18: [Max pooling] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/e/e9/Max_pooling.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

2.9.4.4 Fully Connected Layer

Αυτά είναι τα τελευταία επίπεδα που χρησιμοποιούνται συνήθως στα CNNs. Στα σημεία που τοποθετείται αυτό το επίπεδο, έχουν ήδη περαστεί τα φίλτρα πάνω από τις εικόνες και έχουν σμικρυνθεί αρκετά ώστε να μην εκτοξευτεί ο αριθμός των παραμέτρων που πρέπει να μάθει το δίκτυο. Ο λόγος που συμβαίνει αυτό είναι επειδή σε αντίθεση με τα convolutional επίπεδα, εδώ υπάρχουν συνδέσεις μεταξύ όλων των νευρώνων αυτού και του επόμενου επιπέδου.

Τα αποτελέσματα αυτού του επιπέδου περνιούνται σε μια λίστα. Τα στοιχεία της που έχουν μεγάλες τιμές μας «δείχνουν» ότι αυτά είναι σημαντικά όταν η εικόνα είναι μιας συγκεκριμένου κατηγορίας.

Συνήθως χρησιμοποιούνται ένα ή δύο τέτοιου είδους επίπεδα πριν την τελική κατηγοριοποίηση των εικόνων μας.

2.9.4.5 Batch Normalization

Αυτού του τύπου τα επίπεδα, χρησιμοποιούνται για να ομαλοποιηθούν οι ενεργοποιήσεις της εισόδου πριν αυτές περαστούν στο επόμενο επίπεδο. Με αυτόν τον τρόπο προσπαθούμε να αποφύγουμε απότομες μεγάλες αλλαγές στα βάρη μεταξύ νευρώνων που μπορεί να καθυστερήσουν την εκπαίδευση του νευρωνικού δικτύου.

Αυτά τα επίπεδα μπαίνουν πριν οι πιο συνηθέστερα μετά τα activation επίπεδα και σε πολλές περιπτώσεις μπορούν να βοηθήσουν την πιο γρήγορη εκπαίδευση των δικτύων.

2.9.4.6 Dropout Layers

Αυτού του είδους τα επίπεδα, βοηθάνε στην ομαλοποίηση (regularization) της εκπαίδευσης, εμποδίζοντας το overfitting. Τα dropout επίπεδα μπαίνουν συνήθως μετά από τα πλήρως συνδεδεμένα επίπεδα, επειδή εκεί είναι πιο πιθανό πέσουμε σε overfitting λόγω του πολύ μεγάλου αριθμού παραμέτρων (συνδέσεων) μεταξύ των επιπέδων.

Αυτό που κάνουν είναι τυχαία να μηδενίζουν κάποιο ποσοστό από τα βάρη που συνδέονται στο επόμενο επίπεδο. Παρόλο που μπορεί να μην είναι προφανές πως μια τέτοιου είδους τυχαιότητα μπορεί να βοηθήσει το δίκτυο, τα dropout επίπεδα είναι αρκετά διαδεδομένα.

Πρακτικά «πιέζουν» το δίκτυο να μην βασίζεται σε κάποιες λίγες συνδέσεις για να πάρει απόφαση σχετικά με την κατηγορία μιας εικόνας, αλλά να προσπαθήσει να

«απλώσει» τις τιμές σε πολλαπλούς νευρώνες, άρα να ψάχνει για πολλαπλά χαρακτηριστικά κοινά χαρακτηριστικά που θα έχει μία κατηγορία εικόνων.

2.10 Optimizers, Loss functions και μετρικές

Εφόσον έχει χτίσει όλο το δίκτυο, πρέπει με κάποιο τρόπο να φανεί αν λειτουργεί «καλά». Για αυτό το λόγο γίνεται χρήση των loss functions (συναρτήσεις λάθους).

2.10.1 Loss functions

Με αυτού του είδους τις συναρτήσεις βλέπουμε πόσο «καλές» ή «κακές» προβλέψεις κάνει το δίκτυό. Όσο πιο μικρή η τιμή που γυρίζει η συνάρτηση τόσο καλύτερα αναγνωρίζει το δίκτυο σε ποια κλάση ανήκει το κάθε δεδομένο εισόδου. Όσο πιο μεγάλο το αποτέλεσμα, τόσο περισσότερες αλλαγές θα χρειαστεί να γίνουν στο «εσωτερικό» του δικτύου για να αυξηθεί η ακρίβεια.

Για να αυξηθεί η ακρίβεια, θα πρέπει να αλλάζουν οι τιμές στα βάρη των νευρώνων, κάνοντας χρήση των optimizers. Ιδανικά, το δίκτυο πρέπει σε κάθε εποχή να μειώνει το αποτέλεσμα αυτής της συνάρτησης, ώστε να φαίνεται ότι δουλεύει όλο και καλύτερα.

Υπάρχουν πολλαπλές τέτοιες συναρτήσεις έτοιμες για χρήση μέσα στο κώδικα, από τη mean squared error και hinge, αλλά η πιο δημοφιλής επιλογή τόσο σε προβλήματα κατηγοριοποίησης 2 κλάσεων αλλά και σε περισσότερα είναι η cross entropy.

Με το cross entropy μπορεί να μετρηθεί η απόδοση του μοντέλου όταν έχει ως έξοδο τιμές από το 0 έως το 1. Για αυτό χρησιμοποιείται σχεδόν πάντα μαζί τη σιγμοειδή συνάρτηση ενεργοποίησης στο τελευταίο επίπεδο του δικτύου, για προβλήματα με 2 κλάσεις, και με την softmax, για προβλήματα παραπάνω κλάσεων.

Το λάθος αυξάνεται όσο η πιθανότητα που προβλέφθηκε «απομακρύνεται» από το αναμενόμενο αποτέλεσμα. Ένα τέλειο μοντέλο θα έχει λάθος 0.

2.10.2 Optimizers

Οι loss functions μπορούν μόνο να δείξουν πόσο καλά έχει εκπαιδευτεί ένα δίκτυο, αλλά δεν μπορούν από μόνες τους να το βελτιώσουν. Αυτό πρέπει να το κάνουν οι αλγόριθμοι βελτιστοποίησης.

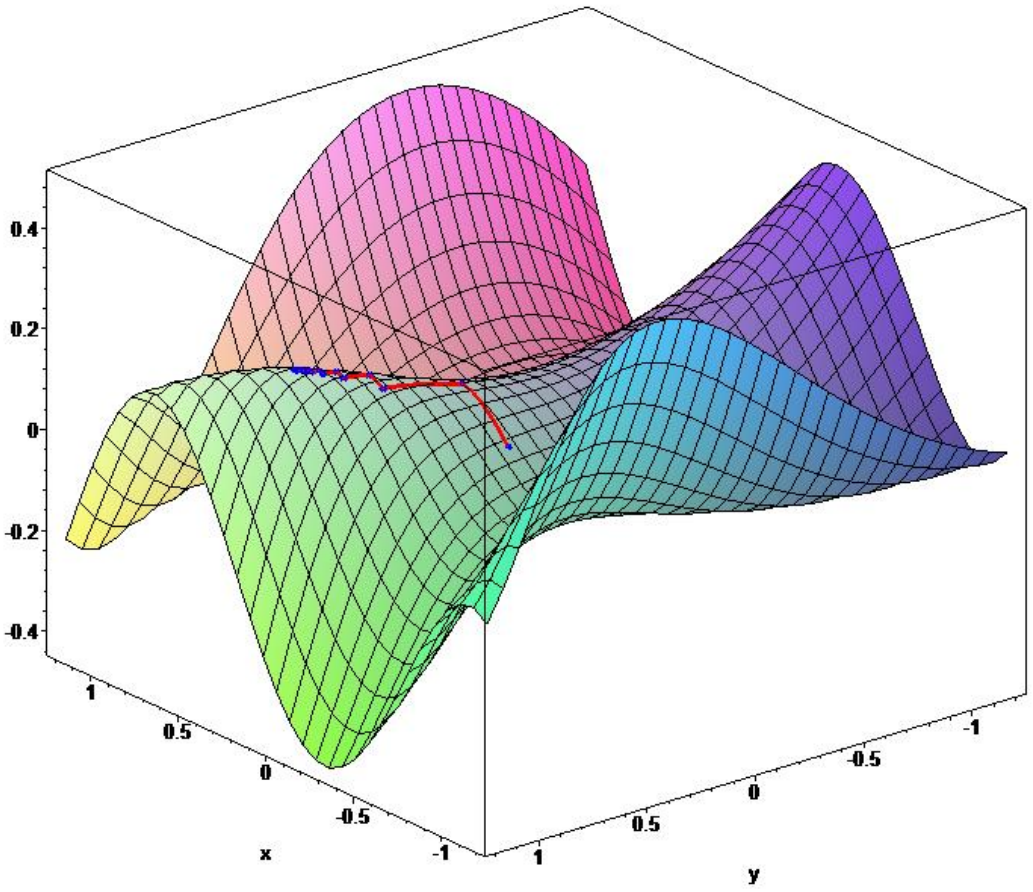
Αν και τα προβλήματα που καλούνται να λύσουν δεν είναι απαραίτητα κοίλα (convex), αντιμετωπίζονται ως τέτοιου είδους. Όταν μία συνάρτηση είναι κοίλη, τότε το

ελάχιστο που θα βρεθεί, θα είναι το ολικό ελάχιστο. Παρόλο που το πρόβλημά μπορεί να μην είναι κοίλο, μέσω αυτών των αλγορίθμων παράγονται αρκετά καλά αποτελέσματα.

Επίσης, σε προβλήματα πολύ υψηλών διαστάσεων, έχει παρατηρηθεί ότι είναι εξαιρετικά δύσκολο να βρεθούμε σε σημείο όπου όλοι οι παράμετροι να είναι σε τοπικό βέλτιστο. Αλλά είναι εξαιρετικά πιθανό να βρεθούμε σε μεγάλες πεδιάδες που ίσως χρειαστεί μεγάλο ρυθμό μάθησης για τις «ξεπεράσουμε».

Ένας από τους πιο γνωστούς αλγορίθμους που χρησιμοποιείται εδώ είναι ο στοχαστικός αλγόριθμος απότομης καθόδου (stochastic gradient descent). Ο αλγόριθμος αυτός αποτελεί μία παραλλαγή του αλγορίθμου απότομης καθόδου, που αντί να ανανεώνει τις τιμές των παραμέτρων μία φορά σε μία εποχή, τις ανανεώνει πολλαπλές φορές σε μικρότερες ομάδες δεδομένων.

Αν και έχει κάποια μειονεκτήματα, έχει αποδειχθεί ότι εκπαιδεύει τα δίκτυα πιο γρήγορα, χωρίς να επηρεάζει αρνητικά την ακρίβεια(Sra et al., 2012).



Εικόνα 2-19: [Αλγόριθμος απότομης καθόδου] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/6/68/Gradient_ascent_%28surface%29.png > [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Υπάρχουν αρκετές παραλλαγές αυτού του αλγορίθμου, οι οποίες προσπαθούν να βελτιώσουν την απόδοση, όπως η επιτάχυνση Nesterov, αλλάζοντας με πιο ευφυείς τρόπους τον ρυθμό μάθησης (learning rate).

Άλλοι αλγόριθμοι που χρησιμοποιούνται αρκετά είναι ο Adam και ο RMSProp και έχουν σαν πλεονέκτημα ότι δεν χρειάζονται, απαραίτητα, να δηλώσουμε κάποιο ρυθμό μάθησης ή να τον μειώνουμε κατά τη διάρκεια της εκπαίδευσης.

2.10.3 Μετρικές

Πολλές φορές χρειάζονται κάποιες έξτρα συναρτήσεις, ώστε να φανεί πιο εύκολα και γρήγορα η απόδοση του δικτύου. Για παράδειγμα μία απλή μετρική είναι η ακρίβεια (accuracy) μέσω της οποία μπορούμε να δούμε το ποσοστό των παραδειγμάτων που κατηγοριοποιήθηκαν σωστά ή να φτιάξουμε και τη μήτρα σύγχυσης ώστε να δούμε ποιες κατηγορίες είναι προβληματικές.

2.11 Overfitting και underfitting

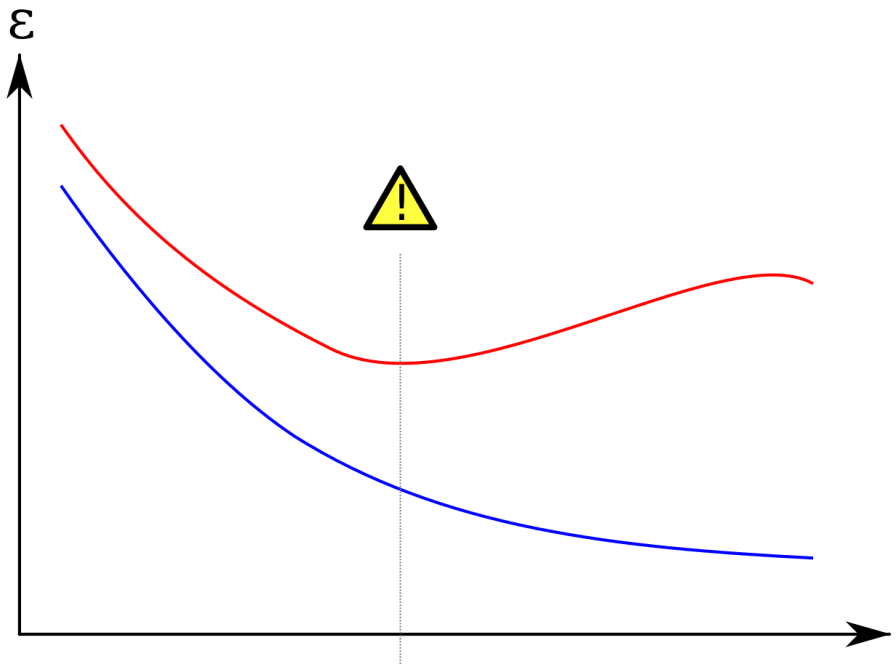
Ακόμα δύο έννοιες που πρέπει να καλυφθούν πριν ξεκινήσει η παρουσίαση της δημιουργίας του νευρωνικού δικτύου είναι αυτές του overfitting (υπερπροσαρμογή) και του underfitting.

Το underfitting συμβαίνει όταν ένα μοντέλο δεν μπορεί να πετύχει αρκετά χαμηλό λάθος (loss) στα δεδομένα εκπαίδευσης. Σε αυτές τις περιπτώσεις, το μοντέλο, αποτυγχάνει να μάθει τα σωστά χαρακτηριστικά των δεδομένων.

Στην αντίπερα όχθη έχουμε το overfitting, όπου το μοντέλο έχει «προσαρμοστεί» πάρα πολύ πάνω στα δεδομένα εκπαίδευσης και δεν γενικεύει καλά σε καινούρια δεδομένα.

Σε κάθε μοντέλο θέλουμε να καταφέρνουμε να κρατάμε το λάθος αρκετά χαμηλά, και παράλληλα να κρατάμε το λάθος των δεδομένων εκπαίδευσης και των δεδομένων επαλήθευσης σε παρόμοια επίπεδα.

Σε περιπτώσεις που το μοντέλο δεν πετυχαίνει αρκετά χαμηλό λάθος μπορούμε να αυξήσουμε των αριθμών των κρυφών επιπέδων ή τον αριθμό των νευρώνων στα ήδη υπάρχοντα. Ενώ μπορούμε να κάνουμε το αντίθετο σε περιπτώσεις overfitting.



Εικόνα 2-20: [Overfitting και underfitting] [ηλεκτρονική εικόνα] Διαθέσιμη: <https://upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Overfitting_svg.svg/1280px-Overfitting_svg.svg.png> [Πρόσβαση στις 13 Σεπτεμβρίου 2019]

Σε περιπτώσεις overfitting μπορούμε επίσης να κάνουμε χρήση και άλλων τεχνικών για να αντιμετωπιστεί, όπως η αύξηση των δεδομένων εκπαίδευσης (ακόμα και τεχνητά, περιστρέφοντας τις εικόνες για παράδειγμα) και μέσω της χρήσης των επιπέδων dropout.

2.12 Διαχωρισμός δεδομένων

Για να μπορέσει να μετρηθεί η «ποιότητα» του μοντέλου που έχει παραχθεί από τα δεδομένα, πρέπει να διαχωριστούν σε τρεις ξεχωριστές κατηγορίες. Τα δεδομένα εκπαίδευσης, τα δεδομένα επαλήθευσης (validation) και τα δεδομένα εξέτασης (test).

Τα δεδομένα εκπαίδευσης, χρησιμοποιούνται ως βάση για την δημιουργία του μοντέλου, και χρησιμοποιούνται για να «μάθει» τα βάρη μεταξύ των νευρώνων και είναι η πρώτη κατηγορία που πρέπει να μπορεί να πετύχει μια αρκετά καλή απόδοση. Για να αποφευχθούν πιθανές περιπτώσεις όπου τα δεδομένα μας είναι ταξινομημένα με κάποιον τρόπο (χαρακτηριστικό) μία καλή τακτική είναι να επιλέγονται τυχαία τα παραδείγματα που θα χρησιμοποιηθούν.

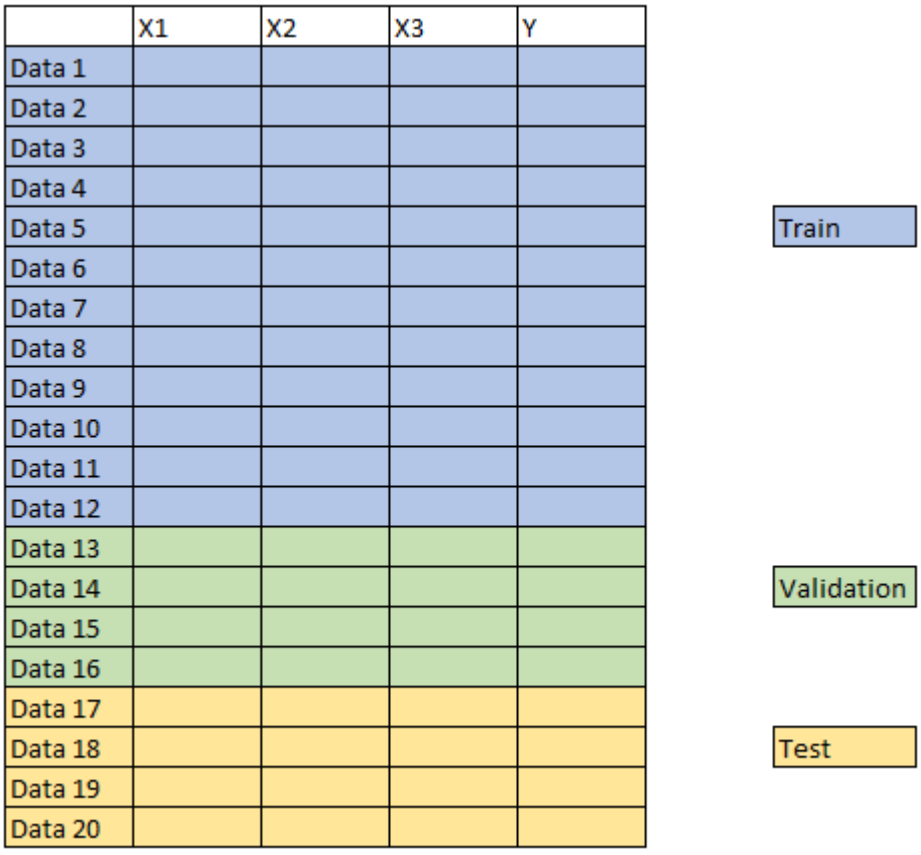
Ο λόγος για τον οποίο δεν πρέπει να συμπεριλαμβάνονται όλα τα δεδομένα σε αυτή την κατηγορία είναι για να φανεί αν το δίκτυό μπορεί να γενικεύσει σωστά ή αν

έχει υπερπροσαρμοστεί. Για να ελεγχθούν αυτές οι υποψίες, γίνεται χρήση της επόμενης κατηγορίας, τα δεδομένα επαλήθευσης.

Με αυτά τα δεδομένα ελέγχεται πόσο καλά, γενικεύει το μοντέλο και αν χρειάζεται οποιουδήποτε είδους αλλαγές, πριν προσπαθήσουμε να τρέξουμε το μοντέλο πάνω στα δεδομένα εξέτασης.

Ένα λάθος που πρέπει να αποφεύγεται είναι να χρησιμοποιηθούν τα δεδομένα εξέτασης για στη δημιουργία ενός μοντέλου και σύμφωνα με εκείνα τα αποτελέσματα, να γίνουν αλλαγές στο στήσιμο του δικτύου. Κάτι τέτοιο δεν πρέπει να συμβαίνει, καθώς με αυτόν τον τρόπο ουσιαστικά χρησιμοποιούνται και τα δεδομένα εξέτασης σαν εκπαίδευσης, που μπορεί να οδηγήσει σε χαμηλή ακρίβεια του μοντέλου σε πραγματικά καινούρια δεδομένα.

Ο τρόπος με τον οποίο χωρίζουμε τα δεδομένα μας επηρεάζεται πάρα πολύ από τον όγκο των δεδομένων που χρησιμοποιούμε.



Εικόνα 2-21: Διαχωρισμός δεδομένων

Ο πιο κοινά αποδεκτός τρόπος για να χωρίσουμε τα δεδομένα μας στις τρεις αυτές κατηγορίες όταν δεν είναι πάρα πολλά τα δεδομένα μας (λιγότερα από δεκάδες χιλιάδες) είναι το 60-20-20, δηλαδή το 60% των δεδομένων θα χρησιμοποιηθούν για την

εκπαίδευση του μοντέλου, το επόμενο 20% θα έχει τον ρόλο των δεδομένων επικύρωσης και το τελευταίο 20% θα βοηθήσει στις δοκιμές του παραγμένου μοντέλου. Αν όμως για κάποιο λόγο, δεν θέλουμε να δημιουργήσουμε δεδομένα τεκμηρίωσης, τότε σύνηθες χωρισμοί είναι το 70-30 ή 80-20.

Σε περιπτώσεις όμως πολύ βαθιάς μάθησης που υπάρχει πληθώρα δεδομένων για τη δημιουργία του μοντέλου, τα ποσοστά των validation και test δεδομένων πέφτουν κοντά στο 10% για το καθένα, καθώς μερικές χιλιάδες δεδομένα είναι αρκετά για αυτούς τους σκοπούς αλλά επίσης πρέπει να μεγιστοποιηθεί ο αριθμός των δεδομένων εκπαίδευσης για την καλύτερη απόδοση του δικτύου.

3 Περιγραφή Προβλήματος

3.1 Επιλογή δεδομένων

Μία από τις πιο γνωστές ιστοσελίδες για άτομα που ασχολούνται με την επιστήμη δεδομένων (data science) είναι το (“Kaggle: Your Home for Data Science,” n.d.). Το kaggle είναι μια ιστοσελίδα που προσφέρει πολλές διαφορετικές λειτουργίες όπως η εύκολη εύρεση δεδομένων από άλλους χρήστες που μπορούν να χρησιμοποιηθούν για την εκπαίδευση νευρωνικών δικτύων, αλλά και η συμμετοχή σε διάφορους διαγωνισμούς (μερικές φορές με χρηματικά έπαθλα), από ινστιτούτα ή οργανισμούς, που προσπαθούν να λύσουν προβλήματα της καθημερινότητας, ελκύοντας μια ευρεία γκάμα επαγγελματιών από κάθε γωνιά του πλανήτη.

Για αυτό το λόγο επιλέχθηκε ένα σετ δεδομένων από έναν ενεργό διαγωνισμό (τη στιγμή που ξεκίνησε η συγγραφή αυτής της διπλωματικής εργασίας). Θα χρησιμοποιηθεί το dataset από τον διαγωνισμό Plant Seedlings Classification(“Plant Seedlings Classification,” n.d.). Δεν προτιμήθηκε η χρήση ενός από τα κλασσικά dataset όπως του MNIST Digit (“MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges,” n.d.), ή του MNIST Fashion(“Fashion MNIST,” n.d.) dataset επειδή τα δεδομένα είναι εξαιρετικά «καθαρά», πράγμα που σημαίνει ότι δεν χρειάζεται κάποια προ-επεξεργασία αλλά και είναι πολύ εύκολο με ένα απλό CNN να επιτευχθεί ακρίβεια κοντά στο 100 %, μέσα σε λίγες εποχές.

Από την άλλη πλευρά απορριφθήκανε dataset που δεν επιτυγχάνεται με ευκολία μεγάλη ακρίβεια, όπως το ImageNet(“ImageNet,” n.d.), ή διαγωνισμούς που είναι ανοιχτοί από την Google όπως το Google Landmark Recognition 2019(“Google Landmark Recognition 2019,” n.d.) καθώς τα δεδομένα αυτά είναι περιλαμβάνουν έναν εξαιρετικά μεγάλο αριθμό εικόνων, 155 GB και 500 GB αντίστοιχα, πράγμα που θα καθιστούσε την εκπαίδευση ενός «βαθιού» δικτύου απαγορευτική για ένα σύστημα με μόνο μία κάρτα γραφικών.

Το dataset που επιλέχθηκε είναι κοντά στα 2 GB και περιέχει 12 διαφορετικές κατηγορίες φυτών. Η κάθε κατηγορία εμπεριέχει εικόνες από αυτό το φυτό σε διαφορετικά στάδια της ανάπτυξής τους. Έχουν δοθεί τα δεδομένα εκπαίδευσης καθώς και κάποιες φωτογραφίες που θα χρησιμοποιηθούν για πρόβλεψη της κλάσης τους, ανεβάζοντας ένα αρχείο csv στο Kaggle με το όνομα του κάθε αρχείου και κλάση που πιστεύει το δίκτυό ότι ανήκει.

3.2 Δυσκολίες που αναμένονται

Αρχικά η πρώτη δυσκολία έγκειται στο ότι, σε αντίθεση με πολλούς άλλους διαγωνισμούς που προσφέρονται στο kaggle, οι 12 κατηγορίες των δεδομένων δεν έχουν την ίδια κατανομή. Αυτό σημαίνει ότι κάποιες από τις κατηγορίες έχουν αρκετά διαφορετικό αριθμό δεδομένων σε σχέση με κάποιες άλλες, οι πιο ακραίες περιπτώσεις είναι αυτές που οι πιο «δημοφιλείς» κατηγορίες έχουν τρεις φορές περισσότερα δεδομένα σε σχέση με τις λιγότερο «δημοφιλείς». Οπότε είτε θα χρειαστεί να χρησιμοποιηθούν όλα τα δεδομένα ελπίζοντας ότι η κατανομή των δεδομένων πάνω στα οποία θα γίνουν οι προβλέψεις θα είναι παρόμοια ή θα χρησιμοποιηθεί μέρος αυτών για να αποφύγουμε το bias.

Η δεύτερη δυσκολία, η οποία είναι ακόμα πιο ασυνήθιστη, βρίσκεται στο ότι όλες οι εικόνες του διαγωνισμού έχουν διαφορετικό μέγεθος / διαστάσεις. Τα νευρωνικά δίκτυα απαιτούν την ομοιομορφία στα δεδομένα, πράγμα που στην προκειμένη περίπτωσή σημαίνει ότι όλες οι διαστάσεις των αρχείων εκπαίδευσης αλλά και τα τελικά αρχεία πάνω στα οποία θα «ελεγχθεί» το τελικό μοντέλο, πρέπει να έχουν όλα τις ακριβώς ίδιες διαστάσεις. Άρα θα χρειαστεί να γίνει επιλογή μίας ανάλυσης που δεν θα αλλοιώσει πάρα πολύ είτε τις πιο μικρές εικόνες ή τις εξαιρετικά μεγαλύτερες.

Η τρίτη δυσκολία εμφανίζεται με έναν πιο «κοντινό» έλεγχο των δεδομένων. Οι εικόνες έχουν αρκετά διαφορετικό φόντο, με κάποιες από αυτές να έχουν πέτρες, κάποιες να έχουν πλαστικά και τέλος σε κάποιες εικόνες η θέση των φυτών δεν είναι κεντραρισμένη, με τα φυτά να είναι στην άκρη της εικόνας και φυσικά οι εικόνες σε κάθε κατηγορία είναι από τα διαφορετικά στάδια της ανάπτυξης τους που προσθέτει ένα ακόμα επίπεδο δυσκολίας.

Εικόνα 3-1: Διαφορετικές εικόνες τις ίδιας κατηγορίας

3.3 Στήσιμο περιβάλλοντος

Στα πλαίσια αυτής της διπλωματικής θα χρησιμοποιηθούν διάφορα έτοιμα πακέτα κώδικα που έχουν υλοποιηθεί για την καλύτερη δυνατή απόδοση από προγραμματιστικής πλευράς (κάποια από αυτά είναι προγραμματισμένα σε C++ ώστε να το πετύχουν αυτό), αλλά και για διευκόλυνση στην δημιουργία νευρωνικών δικτύων από το μηδέν.

3.3.1 Γλώσσα Προγραμματισμού

Αρχικά γίνεται χρήση της διανομής Anaconda της προγραμματιστικής γλώσσας Python. Η Anaconda είναι μια δωρεάν και ανοιχτού κώδικα διανομή της Python με πολλά προ-εγκατεστημένα πακέτα που χρησιμοποιούνται κατά κύριο λόγο από αναλυτές δεδομένων. Μεταξύ άλλων είναι και το πακέτο numpy που βοηθάει στην ευκολότερη διαχείριση πινάκων υψηλών διαστάσεων, κάτι που είναι εξαιρετικά χρήσιμο για τους σκοπούς που θέλουμε να πετύχουμε.

Η εγκατάσταση της Anaconda γίνεται εύκολα (έχει ακόμα και γραφικό περιβάλλον) από το website: https://www.anaconda.com/ με εκδόσεις τόσο για Windows αλλά και macOS και Linux. Η έκδοση που χρησιμοποιήθηκε κατά τη διάρκεια της συγγραφής της διπλωματικής είναι η 2019.03 και περιείχε την 3,7 έκδοση της Python.

Μέσα από την Anaconda μπορούν να δημιουργηθούν πολλαπλά περιβάλλοντα, το οποίο είναι εξαιρετικό βολικό καθώς πολλά πακέτα της Python, και ιδιαίτερα για πακέτα που θα χρησιμοποιηθούν για τη δημιουργία των νευρωνικών δικτύων, όπως το TensorFlow, μπορούν και συνεργάζονται μόνο με συγκεκριμένες εκδόσεις άλλων πακέτων ή ακόμα και με συγκεκριμένες εκδόσεις της Python. Μέχρι και την προηγούμενη έκδοση του TensorFlow (1.12) η έκδοση Python που υποστήριζε ήταν η 3.6, παρόλο που η τελευταία έκδοση της ήταν η 3.7 εδώ και αρκετούς μήνες.

3.3.2 TensorFlow

Το TensorFlow είναι μια βιβλιοθήκη ανοιχτού κώδικα που χρησιμοποιείται για αριθμητικούς υπολογισμούς. Αναπτύχθηκε από μία ομάδα προγραμματιστών μέσα στη Google αρχικά για εσωτερική χρήση, αλλά το 2015 έγινε η δημόσια διανομή του και από τότε πολλές μεγάλες εταιρίες, στο χώρο της πληροφορικής(“TensorFlow,” n.d.) (και όχι μόνο) κάνουν χρήση της.

Προσφέρονται δύο διαφορετικές εκδόσεις του TensorFlow, η απλή, όπου όλοι οι υπολογισμοί τρέχουν στην κεντρική μονάδα επεξεργασίας (CPU) όπου και μια απλή

εγκατάσταση του πακέτου μέσω του pip (ο package manager της Python) μπορεί ο οποιοσδήποτε να ξεκινήσει να δημιουργεί νευρωνικά δίκτυα πολύ γρήγορα και εύκολα. Το μειονέκτημα είναι ότι ακόμα και οι επεξεργαστές τελευταίας γενιάς (προς οικιακή χρήση) έχουν το μέγιστο 16 με 32 threads, που μπορεί να είναι αρκετοί ακόμα και για επαγγελματική χρήση σε εφαρμογές video editing, αλλά στα πλαίσια της μηχανικής μάθησης μόνο εξαιρετικά μικρά μοντέλα θα μπορέσουν να εκπαιδευτούν σε λογικά χρονικά πλαίσια.

Για αυτόν το λόγο υπάρχει και μια διαφορετική έκδοση, η οποία κάνει χρήση της κάρτας γραφικών (GPU) του υπολογιστή. Σε αντίθεση με τα 16 ή τα 32 threads, ενός κορυφαίου επεξεργαστή τελευταίας γενιάς, οι περισσότερες κάρτες γραφικών τελευταίας γενιάς έχουν πάνω από χίλιους (1000) πυρήνες στην διάθεσή του και συγκεκριμένα αυτή που θα χρησιμοποιήσω εγώ (Nvidia GeForce 2080RTX) έχει 2944 πυρήνες, με 8 GB μνήμης, που θα βοηθήσει στη γρήγορη εκπαίδευση νευρωνικών δικτύων με λίγα κρυφά επίπεδα και σχετικά μικρό αριθμό αρχικών δεδομένων (όπως αυτά που θα χρησιμοποιηθούν από το Kaggle).

Η έκδοση του TensorFlow που χρησιμοποιήθηκε, είναι η 1.13, και είναι η τελευταία διαθέσιμη σταθερή έκδοση, τη στιγμή που γράφεται αυτή η διπλωματική.

3.4 Πρόσθετες βιβλιοθήκες

3.4.1 CUDA

Καθώς θα κάνω χρήση της έκδοσης που στοχεύει στις κάρτες γραφικών θα χρειαστούν και έξτρα βιβλιοθήκες και λογισμικό όπως το CUDA toolkit για τις κάρτες της Nvidia. Εδώ χρειάζεται έξτρα προσοχή ανάλογα με την έκδοση του TensorFlow που θα χρησιμοποιηθεί, ποια έκδοση CUDA υποστηρίζεται (πάντα οι εκδόσεις της CUDA «τρέχουν» περισσότερο από τις υποστηριζόμενες), η τελευταία έκδοση 1.13 του TensorFlow ξεκίνησε να υποστηρίζει επίσημα την CUDA 10 (μέχρι πρότινος ήταν η έκδοση 9). Επίσης, χρειάζεται προσοχή, ώστε να εξεταστεί και ποια έκδοση CUDA υποστηρίζει η κάθε κάρτα γραφικών, οι κάρτες της σειράς 20 της Nvidia δεν υποστήριζαν την CUDA 9 που σήμαινε ότι έπρεπε να γίνει έξτρα διαδικασία ώστε το TensorFlow να μπορεί να τρέξει «σωστά» σε αυτές τις κάρτες.

3.4.2 Keras

Παρά το γεγονός ότι το ίδιο το TensorFlow έχει κάνει το προγραμματισμό των νευρωνικών δικτύων πολύ απλό, έχουν βγει και άλλες βιβλιοθήκες (πακέτα), τα οποία χρησιμοποιούν ως βάση κάποια τέτοιου είδους υλοποίηση (TensorFlow, Theano κλπ) αλλά έχουν καταφέρει να απλοποιήσουν ακόμα περισσότερο την όλη διαδικασία.

Μία τέτοια βιβλιοθήκη είναι το keras, η οποία θα χρησιμοποιηθεί κατά κόρον στις υλοποιήσεις. Αυτό γίνεται για δύο βασικούς λόγους, αρχικά δεν χρειάζεται κάποια έξτρα εγκατάσταση για τη χρήση του keras, καθώς το TensorFlow ήδη έχει συμπεριλάβει τις περισσότερες δυνατότητές του και έχουμε πρόσβαση μέσω imports όπως tensorflow.python.keras.

Ο δεύτερος λόγος για τον οποίο χρησιμοποιήθηκε το keras είναι επειδή τη στιγμή που γράφεται αυτή η εργασία, η έκδοση 2.0 του TensorFlow βρίσκεται σε έκδοση alpha και μεταξύ των πολλών αλλαγών στο api του, έχει ανακοινωθεί ο τρόπος δημιουργίας των δικτύων μέσω του keras θα γίνει ο κεντρικός, άρα δεν υπάρχει πλέον λόγος να γραφεί ο πιο «πολύπλοκος» κώδικας από το βασικό πακέτο καθώς σε μερικούς μήνες θα θεωρείται «παρωχημένος».

3.4.3 Hyperopt και Hyperas

Όπως θα περιγραφεί και πιο κάτω, η δημιουργία του «ιδανικού» μοντέλου για τα δεδομένα, βασίζεται στις πολλαπλές υλοποιήσεις όπου γίνεται αναζήτηση των παραμέτρων που θα εκπαιδεύσουν το δίκτυο όσο καλύτερα γίνεται.

Επειδή όμως οι παράμετροι που πρέπει να επιλεχθούν, είναι πάρα πολλοί, από τον αριθμό των κρυφών επιπέδων και τη συνάρτηση λάθους, ως τον αριθμό των φίλτρων σε κάθε convolutional επίπεδο, είναι πιο εύκολο να χρησιμοποιούμε εργαλεία που επιτρέπουν το τρέξιμο πολλαπλών μοντέλων, διαδοχικά, και να επιστρέφουν την καλύτερη υλοποίηση.

Κάποια από αυτά είναι το Hyperopt και η «επέκτασή» του το Hyperas. Μέσω αυτών των βιβλιοθηκών μπορούν να δοθούν κάποια σετ από επιλογές στο δίκτυο για κάθε παράμετρο που πρέπει να βελτιστοποιηθεί καθώς και ο συνολικός αριθμός των διαφορετικών μοντέλων που μπορούν να εκπαιδευτούν, αλλά και ο αριθμό των εποχών που θα εκπαιδεύεται το κάθε μοντέλο.

3.4.4 Εγκατάσταση TensorFlow

Εν μέσω όλης αυτής της «αναταραχής», η ομάδα του Anaconda είδε ότι το εκρηκτικό ενδιαφέρον της κοινότητας της πληροφορικής και την στροφή στην μηχανική μάθηση από πολλά πεδία και απλοποίησε την διαδικασία μετά τις 7 Σεπτεμβρίου 2018. Πριν από εκείνο το σημείο, η διαδικασία που θα χρειαζόταν τουλάχιστον μία μέρα για το σωστό «στήσιμο» όλων των απαραίτητων εκδόσεων της Python, των βιβλιοθηκών (τόσο της ίδιας αλλά και των εξωτερικών όπως της Nvidia) πλέον μπορεί να ετοιμαστεί μέσα σε 10 λεπτά (που εξαρτάται μόνο από την ταχύτητα σύνδεσης στο Internet).

Χρησιμοποιώντας την εντολή: conda create -n tensorflow_gpuenv tensorflow-gpu, ο package manager της Anaconda (το conda) δημιουργεί ένα καινούριο περιβάλλον με την τελευταία έκδοση, που είναι διαθέσιμη εκείνη τη στιγμή, του Tensorflow και κατεβάζει όλες τις απαραίτητες βιβλιοθήκες αλλά και μερικές προαιρετικές αλλά που χρησιμοποιούνται εξαιρετικά συχνά.

```
(base) C:\Users\Chris>conda create -n tensorflow_gpuenv tensorflow-gpu
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Chris\Anaconda3\envs\tensorflow_gpuenv

  added / updated specs:
    - tensorflow-gpu


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    absl-py-0.7.1              |           py37_0         158 KB
    astor-0.7.1                |           py37_0          44 KB
    certifi-2019.3.9           |           py37_0         155 KB
    gast-0.2.2                 |           py37_0         138 KB
```

Εικόνα 3-2: Δημιουργία περιβάλλοντος TensorFlow

Ακόμα και αν το Anaconda που εγκαταστάθηκε είναι σε διαφορετική έκδοση Python από αυτή που υποστηρίζει το TensorFlow θα εγκαταστήσει αυτή που χρειάζεται ώστε να μπορέσει να χρησιμοποιηθεί σε οποιοδήποτε IDE προτιμάει ο κάθε ερευνητής.

Ένα ακόμα πλεονέκτημα που υποστηρίζει η ομάδα πίσω από το Anaconda ότι έχει η υλοποίηση που προσφέρουν, πέρα από την ευχρηστία, είναι ότι η απόδοση του TensorFlow είναι μερικές φορές καλύτερη από την αντίστοιχη εγκατάσταση μέσω του pip(“TensorFlow in Anaconda,” 2018).

4 Μεθοδολογία

4.1 Προ-επεξεργασία δεδομένων

Πριν οποιοδήποτε προσπάθεια για τη δημιουργία του νευρωνικού δικτύου, πρέπει να ελεγχθεί ο συνολικός αριθμός των δεδομένων εκπαίδευσης αλλά και η κατανομή τους στις διαφορετικές κατηγορίες. Περιπτώσεις όπου μία κλάση έχει πολλά περισσότερα παραδείγματα, μπορούν να οδηγήσουν σε δημιουργία δικτύων τα οποία να είναι τείνουν στην κατηγοριοποίηση δεδομένων, σε αυτήν, λανθασμένα.

4.2 Επιλογή δεδομένων

Αρχικά δημιουργήθηκε μια μέθοδος, που εκτυπώνει στην κονσόλα τον αριθμό των κλάσεων και τον γυρνάει ώστε να χρησιμοποιηθεί αργότερα στη δημιουργία του μοντέλου.

```
def show_number_classes(directory):
    num_classes = len(listdir(directory))
    print('Number of classes:', num_classes)
    return num_classes
```

Εικόνα 4-1: Υπολογισμός αριθμού κλάσεων

Η μέθοδος listdir είναι του πακέτου os και επιστρέφει τη λίστα με τα ονόματα των αρχείων (ή εδώ των φακέλων) για το φάκελο που παίρνει ως όρισμα. Στην προκειμένη περίπτωση επιστρέφει τον αριθμό 12.

Έπειτα κατασκευάστηκε μία μέθοδος που εκτυπώνει την κατανομή των δεδομένων σε κάθε μία από τις κλάσεις σε ένα διάγραμμα.

```
def show_images_chart(directory):
    num_img_per_class = []
    class_names = []
    for root, dirs, files in walk(directory):
        for name in dirs:
            num_img_per_class.append(len(listdir(join(root, name))))
            class_names.append(name)

    numeric_values = np.arange(len(class_names))

    plt.bar(numeric_values, num_img_per_class, align='center', alpha=0.5)
    plt.xticks(numeric_values)
    plt.ylabel('Number of images')
    plt.xlabel('Categories')
    plt.title('Distribution of images')
    plt.show()
    return np.array(num_img_per_class).min()
```

Εικόνα 4-2: Δημιουργία γραφήματος αριθμού εικόνων ανά κλάση

Εδώ χρησιμοποιήθηκε και μία άλλη μέθοδο του os, το walk μέσω της οποίας μπορούμε να «περπατήσουμε» μέσα στον φάκελο που δίνουμε ως όρισμα αλλά και μέσα στους υπο-φακέλους και επιστρέφονται τα ονόματα όλων των αρχείων που περιέχει ο καθένας τους.

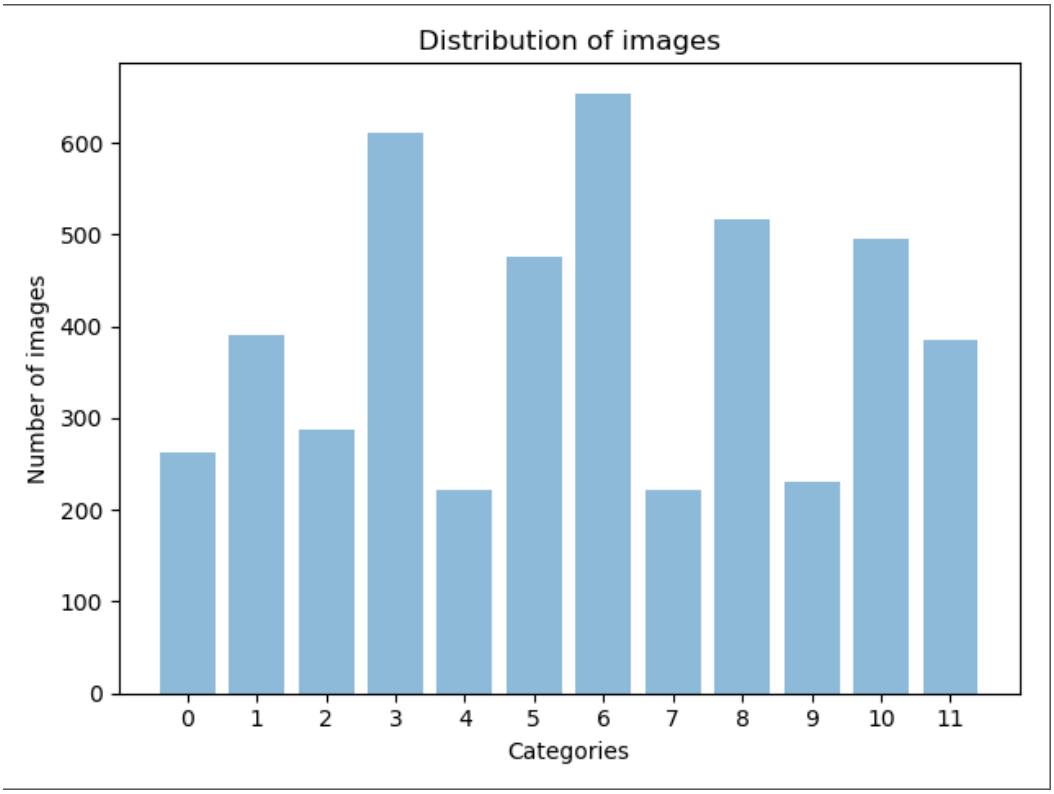
Δημιουργούνται δύο πίνακες, ο ένας με τον αριθμό των δεδομένων εκπαίδευσης της κάθε κλάσης και ο δεύτερος με τα ονόματά τους. Αλλά επειδή τα ονόματα των κλάσεων είναι αρκετά μεγάλα και δεν είναι αναγνώσιμα αν μπούνε μέσα στο διάγραμμα (Black-grass, Common Chickweed) τα ονόματα αυτά, μετατρέπονται σε τιμές από το 0 έως το 11 (καθώς έχουμε 12 κλάσεις), ώστε να χρησιμοποιηθούν αυτές.

Το μοντέλο άλλωστε θα δουλέψει με τις τιμές 0 έως 11 όταν θα χτίζεται και όχι με τις ονομαστικές, οι οποίες είναι μόνο βολικές για τους ανθρώπους.

Τρέχοντας την παραπάνω μέθοδο εμφανίζεται ο πίνακας της κατανομής που δείχνει ότι εξαρχής υπάρχουν κάποια προβλήματα με τα δεδομένα. Συνήθως τα δεδομένα σε τέτοιους διαγωνισμούς είναι «φτιαγμένα» ώστε όλες οι κλάσεις να έχουν παρόμοιο αριθμό δεδομένων.

Κάτι τέτοιο δεν συμβαίνει εδώ και πρέπει αντιμετωπιστεί, ώστε να επιτευχθεί μία καλή απόδοση από τα μοντέλα που θα δημιουργηθούν στη συνέχεια. Μικρές αποκλίσεις (του τύπου 10%) δεν είναι τόσο προβληματικές, αλλά εδώ υπάρχουν 3 κλάσεις με λιγότερα από 200 παραδείγματα ενώ οι 2 κορυφαίες έχουν πάνω από 600, δηλαδή

σχεδόν 3 φορές περισσότερα. Αυτό οδηγεί το μοντέλο να έχει «κλίση» και να θεωρεί ότι πολλά δεδομένα θα ανήκουν στις κλάσεις 3 και 6 επειδή είναι πολλά περισσότερα.



Εικόνα 4-3: Αρχική κατανομή εικόνων

Αυτή την κατάσταση μπορεί να αντιμετωπιστεί διαφορετικούς τρόπους. Μία επιλογή, είναι η προσπάθεια εύρεσης καινούριων δεδομένων, είτε μέσω αναζήτησης στο Internet είτε να «τραβήξουμε» και άλλες φωτογραφίες. Ένα άλλος τρόπος είναι να δημιουργηθούν πλασματικά «καινούρια» δεδομένα, αντιστρέφοντας τις εικόνες οριζοντίως ή καθέτως, να γίνει μεγένθυνση κ.ο.κ.

Ένας τελευταίος τρόπος είναι και αυτός που επιλέχθηκε, είναι να επιλεγούν τόσες τυχαίες εικόνες από κάθε κλάση, όσες έχει η κλάση (ή κλάσεις) με τις λιγότερες. Αυτό θα μειώσει αρκετά τα δεδομένα και μπορεί να έχει αρνητικό αντίκτυπο στη δημιουργία του μοντέλου, αν τύχει να πάρουμε «παρόμοιες» εικόνες από τις κλάσεις με τα περισσότερα δεδομένα, αλλά μπορεί να γίνει τροφοδότηση όλων των δεδομένων στο τελικό μοντέλο, με τα εκπαιδευμένα βάρη ελπίζοντας ότι θα μπορέσουν να γίνουν οι απαραίτητες μικρο-διορθώσεις.

Επίσης αργότερα μπορούν να χρησιμοποιηθούν οι κλάσεις του keras ώστε να αυξηθεί πλασματικά, ο αριθμός των δεδομένων.

Για να επιτευχθεί αυτό το αποτέλεσμα δημιουργήθηκαν οι εξής μεθόδοι: πρώτη είναι η split_data όπου παίρνει σαν ορίσματα τους φακέλους που θα μετακινηθούν τα δεδομένα που θα χρησιμοποιηθούν για εκπαίδευση και επαλήθευση, καθώς και ο ελάχιστος αριθμός δεδομένων από όλες τις κλάσεις (που θα χρησιμοποιηθεί για τον υπολογισμό του αριθμού τον δεδομένων που θα πάνε σε κάθε φάκελο) αλλά και το ποσοστό του διαχωρισμού μεταξύ αυτών.

```
def split_data(source_dir, train_dir, val_dir, min_examples, split):
    files = []
    for filename in listdir(source_dir):
        file = source_dir + filename
        if getsize(file) > 0:
            files.append(filename)
        else:
            print(filename + " is zero length, so ignoring.")

    training_length = int(split * min_examples)
    shuffled_set = random.sample(files, len(files))
    training_set = shuffled_set[0:training_length]
    validation_set = shuffled_set[training_length:min_examples]

    for filename in training_set:
        this_file = source_dir + filename
        destination = train_dir + filename
        copyfile(this_file, destination)

    for filename in validation_set:
        this_file = source_dir + filename
        destination = val_dir + filename
        copyfile(this_file, destination)
```

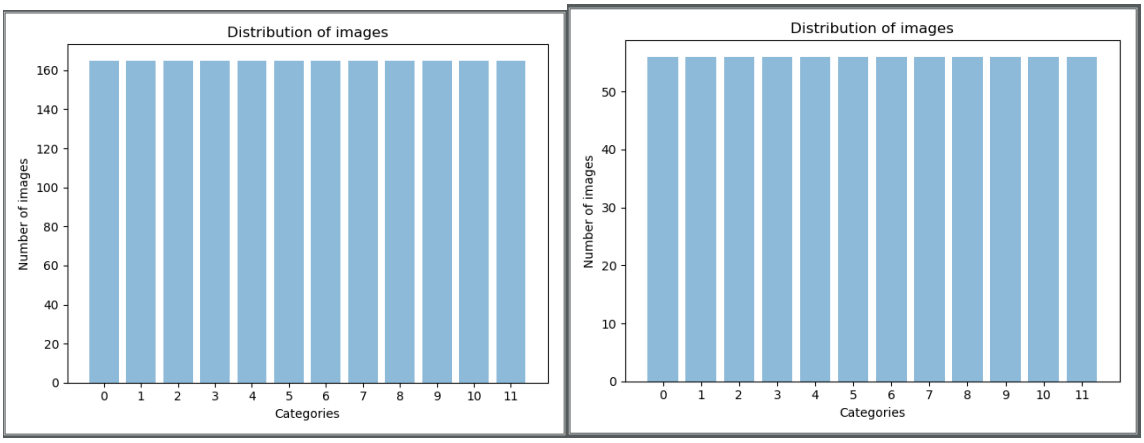
Εικόνα 4-4: Διαχωρισμός δεδομένων

Η δεύτερη μέθοδος (finalise_train_val_data) είναι αυτή που θα φτιάξει τους φακέλους όπου θα χωριστούν τα δεδομένα για κάθε κλάση και θα καλέσει την split_data.

```
def finalise_train_val_data(directory, train_dir_name, val_dir_name, min_img, split):
    for root, dirs, files in walk(directory):
        for name in dirs:
            try:
                mkdir(join(train_dir_name, name) + '/')
                mkdir(join(val_dir_name, name) + '/')
            except OSError:
                pass
            split_data(join(root, name) + '/', join(train_dir_name, name) + '/',
                       join(val_dir_name, name) + '/', min_img, split)
```

Εικόνα 4-5: Μέθοδος δημιουργίας φακέλων

Μετά από όλη αυτή τη διαδικασία, τρέχοντας ξανά τη μέθοδο show_images_chart για τους καινούριους φακέλους εμφανίζεται η καινούρια κατανομή τους.



Εικόνα 4-6: Κατανομή τελικών δεδομένων, με διαχωρισμό 75 – 25

Παρόλο που δεν φτιάχνεται, ρητά μια ακόμα ομάδα για τα δεδομένα ελέγχου (test data), υπάρχουν ήδη τα δεδομένα που έχουν δοθεί από το διαγωνισμό για να προβλεφθεί η κατηγορία τους και σύμφωνα με τα αποτελέσματά τους θα βαθμολογηθεί το μοντέλο.

4.2.1 Οπτικοποίηση αποτελεσμάτων εκπαίδευσης

```
def train_val_plot(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    plt.subplot(1, 2, 1)
    plt.plot(acc, color='b', label="Training Accuracy")
    plt.plot(val_acc, color='r', label="Validation Accuracy")
    plt.title('Training and validation accuracy')
    plt.legend(loc='best', shadow=True)

    plt.subplot(1, 2, 2)
    plt.plot(loss, color='b', label="Training Loss")
    plt.plot(val_loss, color='r', label="Validation Loss")
    plt.title('Training and validation loss')
    plt.legend(loc='best', shadow=True)

    plt.show()
```

Εικόνα 4-7: Οπτικοποίηση αποτελεσμάτων

4.3 Επιλογή παραμέτρων

Λόγω της φύσης του προβλήματος, όπως έχει αναφερθεί και πιο πάνω, δεν είναι δυνατό να γνωρίζουμε εξαρχής ποιες παράμετροι θα δουλέψουν καλύτερα για το συγκεκριμένο dataset, οπότε πρέπει να δημιουργηθούν πολλαπλοί διαφορετικοί συνδυασμοί ώστε να «χτιστεί» σιγά σιγά ένα μοντέλο που να πετυχαίνει την καλύτερη δυνατή απόδοση τόσο στα δεδομένα εκπαίδευσης, όσο και στα δεδομένα επαλήθευσης

Αρχικά θα «τρέξουν» πολλαπλά δίκτυα με λίγες εποχές, π.χ. 10, ώστε να φανεί η συμπεριφορά τους με αυτά τα δεδομένα. Με αυτό τον τρόπο δεν θα σπαταλήσουμε χρόνο εκπαιδεύοντας μεγάλα δίκτυα που μπορεί να αποδειχθούν ότι δεν έχουν καλή απόδοση. Τα αποτελέσματα των δικτύων θα χρησιμοποιηθούν ως ανατροφοδότηση, δείχνοντας τα προβληματικά σημεία στα οποία θα πρέπει να επικεντρωθούμε.

4.4 Δημιουργία πρώτου μοντέλου

Εφόσον έχουν επιλεγεί τα δεδομένα που θα χρησιμοποιηθούν σε κάθε κατηγορία, ξεκινάει το κυρίως κομμάτι, αυτό της δημιουργίας του μοντέλου.

Δυστυχώς δεν υπάρχει ένας σίγουρος τρόπος με τον οποίο αν δημιουργηθεί κάποιο μοντέλο, θα πετύχει τα επιθυμητά αποτελέσματα. Υπάρχουν όμως κάποιες συνηθισμένες αρχιτεκτονικές, με βάση τις οποίες θα δημιουργούνται τα δίκτυα.

Η πρώτη απόπειρα θα περιέχει ένα απλό δίκτυο με ένα Convolutional επίπεδο με 32 φίλτρα, 3 επί 3 με ενεργοποίηση relu και θα δέχεται έγχρωμες εικόνες 128 επί 128 εικονοστοιχείων. Με αυτόν τον τρόπο θα υπάρχει αρκετή πληροφορία μέσα στην εικόνα, ώστε να μπορέσει να γίνει εξαγωγή αρκετά πολύπλοκων χαρακτηριστικών, με πολλαπλά convolutional επίπεδα σε επόμενες προσπάθειες.

Καθώς προς το παρών δεν υπάρχουν πολλά επίπεδα που να μειώνουν την ανάλυση των εικόνων, δεν θα χρησιμοποιηθεί το padding. Στο keras η επιλογή padding valid, σημαίνει ότι δεν θα χρησιμοποιηθεί καθόλου padding, ενώ η επιλογή padding same είναι το zero-padding που περιεγράφηκε και πιο πάνω. Επίσης καθώς δεν υπάρχει δηλωμένο stride, το keras θα πάρει την προκαθορισμένη τιμή 1.

Έπειτα θα χρησιμοποιηθεί το Batch Normalization, που αν και μπορεί να καθυστερήσει την εκπαίδευση του μοντέλου, βοηθάει ώστε να «σταθεροποιηθεί», επιτρέποντάς μας έτσι να επικεντρωθούμε στην επιλογή των παραμέτρων του δικτύου.

Έπειτα χρησιμοποιείται το Flatten, πριν από το πλήρως συνδεδεμένο επίπεδο με 128 παραμέτρους και τέλος στο επίπεδο εξόδου υπάρχει ακόμα ένα πλήρες συνδεδεμένο

layer με παραμέτρους όσες και οι τάξεις και ενεργοποίηση softmax ώστε να γίνει εξαγωγή των αποτελεσμάτων.

```
def create_model(dimension, classes):
    model = Sequential()
    model.add(Conv2D(32, 3, padding='valid', activation='relu', input_shape=(dimension, dimension, 3)))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(classes, activation='softmax'))
    return model
```

Εικόνα 4-8: Αρχικό μοντέλο

Το επόμενο κομμάτι είναι η κλήση του compile ώστε να τεθεί ο optimizer, η loss function αλλά και η μετρική που θα χρησιμοποιεί το μοντέλο.

```
def compile_model(model):
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

Εικόνα 4-9: Επιλογή optimizer, loss function και μετρικής

Ο λόγος για τον οποίο δεν χρησιμοποιείται από την αρχή το Hyperas, είναι επειδή ο αριθμός των παραμέτρων που πρέπει να μάθει αυτό το δίκτυο είναι εξαιρετικά μεγάλος καθώς δεν χρησιμοποιούνται πολλαπλά επίπεδα convolutions για να τον μειώσει. Με μεγάλο αριθμό νευρώνων στο πλήρως συνδεδεμένο επίπεδο, το TensorFlow δεν μπορούσε να βρει αρκετούς πόρους για να τρέξει τα μοντέλα.

Αλλά σε παρακάτω στάδια θα περιγραφεί, η χρήση του, καθώς και ο ιδιαίτερος τρόπος με τον οποίο θα χρειαστεί να «στηθεί» το δίκτυο για να μπορέσει να λειτουργήσει ο κώδικας.

Μετά τη δημιουργία του ίδιου του δικτύου, ετοιμάζονται οι γεννήτριες δεδομένων που προσφέρει το keras, τόσο για τα δεδομένα εκπαίδευσης αλλά και για τα δεδομένα επαλήθευσης. Προς το παρών το μόνο που επιτυγχάνεται είναι η κανονικοποίηση των εικόνων ώστε οι τιμές των εικονοστοιχείων για όλες τις εικόνες να έχουν τιμές από 0 έως 1 και όχι έως 255 όπως έχουν κανονικά.

```
def create_datagens():
    train_datagen = ImageDataGenerator(rescale=1. / 255)

    validation_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    return train_datagen, validation_datagen
```

Εικόνα 4-10: Κανονικοποίηση εικόνων

Το επόμενο κομμάτι έχει να κάνει με τη φόρτωση των δεδομένων στο μοντέλο μας. Το keras προσφέρει εύκολο τρόπο να φορτώθούν κατευθείαν από τους φακέλους του υπολογιστή. Αυτό είναι ιδιαίτερα βολικό για μεγαλύτερα dataset όπου δεν μπορούν να φορτωθούν όλα μέσα στη μνήμη RAM. Μπορεί να επιλεχθεί το μέγεθος του batch,

δηλαδή πόσα δεδομένα θα φορτώνει ανά φορά εκπαίδευσης, αν υπάρχουν πολλαπλές (categorical) κλάσεις ή μόνο 2 (binary) καθώς και την ανάλυση με την οποία θα φορτωθούν στο δίκτυο. Με αυτή την λειτουργικότητα, δεν χρειάζεται η έξτρα δουλειά για την προ-επεξεργασία των δεδομένων, αλλάζοντας το μέγεθός τους.

```
def create_generators(train_datagen, validation_datagen, train_dir, validation_dir, dimension, batch_size):
    train_generator = train_datagen.flow_from_directory(train_dir,
                                                        batch_size=batch_size,
                                                        class_mode='categorical',
                                                        target_size=(dimension, dimension))

    validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                                  batch_size=batch_size,
                                                                  class_mode='categorical',
                                                                  target_size=(dimension, dimension))
    return train_generator, validation_generator
```

Εικόνα 4-11: Φόρτωση δεδομένων με τη μέθοδο create_generators

Το τελευταίο κομμάτι έχει να κάνει με την φόρτωση των generators μέσα στο μοντέλο αλλά και την επιλογή των εποχών εκπαίδευσης, αλλά και την επιλογή των βημάτων ανά εποχή τόσο για τα δεδομένα εκπαίδευσης όσο και για τα δεδομένα επαλήθευσης. Εφόσον φορτώνονται ανά ομάδες (batches) πρέπει να καταφέρουν να περάσουν όλα από μία φορά ανά εποχή μέσα από το μοντέλο.

```
def fit_model(model, train_generator, validation_generator, batch_size, epochs):
    history = model.fit_generator(train_generator,
                                  steps_per_epoch=train_generator.n // batch_size,
                                  epochs=epochs,
                                  validation_data=validation_generator,
                                  validation_steps=validation_generator.n // batch_size)
    return history
```

Εικόνα 4-12: Φόρτωση δεδομένων στο μοντέλο και εποχές εκπαίδευσης

4.4.1 Πρώτο τρέξιμο

Τα δεδομένα εισόδου θα δοθούν αρχικά σε μία αρκετά μικρή διάσταση 128 επί 128 εικονοστοιχείων. Αυτή η ανάλυση επιλέχθηκε για να μπορέσουν να χρησιμοποιηθούν παραπάνω convolutional και MaxPooling επίπεδα σε επόμενα μοντέλα. Ένας άλλος λόγος είναι ότι τα δεδομένα διαφέρουν πάρα πολύ σε διαστάσεις, υπάρχουν εικόνες με ανάλυση 74 επί 74 μέχρι και 1900 επί 1200, οπότε επιλέχθηκε μία διάσταση που θα μπορέσει να τη διαχειριστεί αρκετά εύκολα το μηχάνημα στο οποίο θα τρέχουν τα δίκτυα.

Σαν optimizer χρησιμοποιείται ο Adam που είναι αρκετά διαδεδομένη η χρήση του λόγω σχετικά καλής απόδοσης αλλά επίσης αφαιρεί τη διαδικασία επιλογής έξτρα παραμέτρων, το loss function είναι το categorical crossentropy, και πάλι μία αρκετά δημοφιλής επιλογή που χρησιμοποιείται σε συνδυασμό με τη softmax, με μία απλή μετρική, την ακρίβεια. Δεν δημιουργούνται έξτρα δεδομένα εκπαίδευσης, μόνο κανονικοποιούνται, ενώ η εκπαίδευση θα γίνει για 10 εποχές και θα χρησιμοποιηθούν ομάδες (batches) των 32 τόσο για την εκπαίδευση όσο και την επαλήθευση.

Μέσω της μεθόδους model.summary() μπορεί να γίνει εκτύπωση στην κονσόλα το μοντέλο που έχει φτιαχτεί, καθώς και οι διαστάσεις των εικόνων μετά από κάθε επίπεδο αλλά και τον αριθμό των παραμέτρων που θα χρειαστεί να μάθει το δίκτυο.

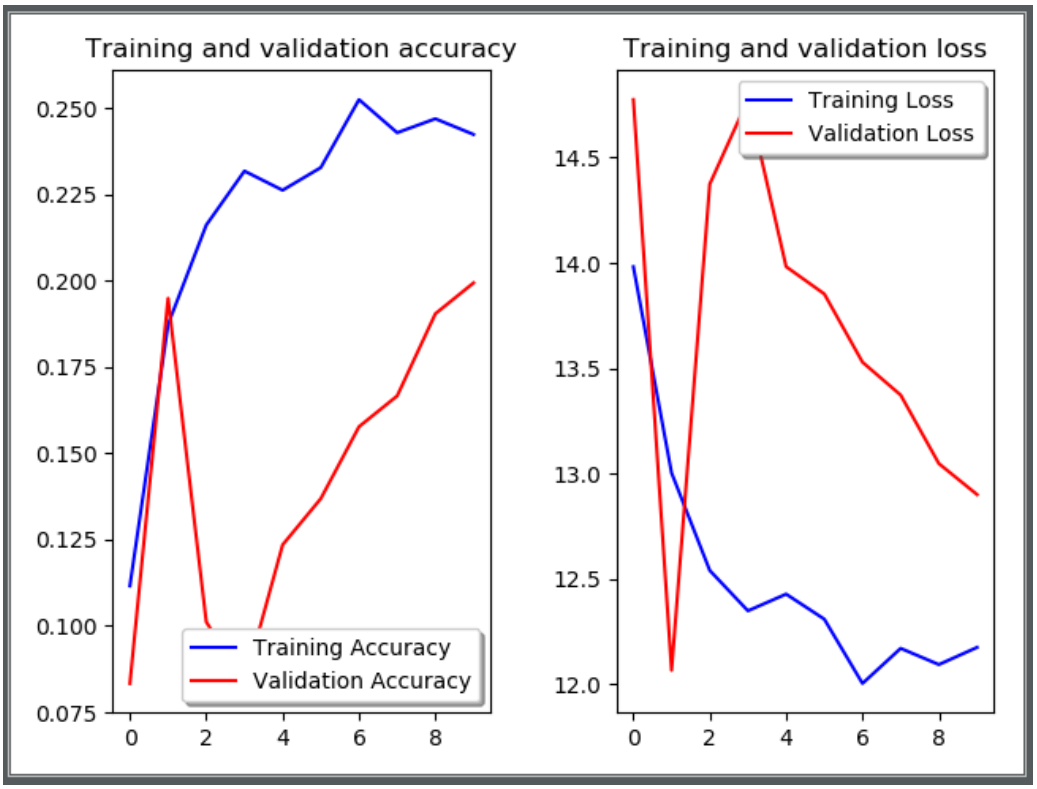
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
batch_normalization_v1 (Batc (None, 126, 126, 32)      128
_________________________________________________________________
flatten (Flatten)            (None, 508032)            0
_________________________________________________________________
dense (Dense)                (None, 128)               65028224
_________________________________________________________________
dense_1 (Dense)              (None, 12)                1548
=================================================================
Total params: 65,030,796
Trainable params: 65,030,732
Non-trainable params: 64
_________________________________________________________________
```

Εικόνα 4-13: Λεπτομέρειες πρώτου δικτύου

Από την παραπάνω εικόνα, φαίνεται πως χωρίς το padding, η ανάλυση των δεδομένων εξόδου από το πρώτο επίπεδο μειώνεται ελαφρώς (126x126) και πως χρειάζεται η εκπαίδευση ενός εξαιρετικά μεγάλου αριθμού παραμέτρων, λίγο πάνω από 65 εκατομμύρια.

4.4.2 Αποτελέσματα πρώτου δικτύου

Τα αποτελέσματα του δικτύου είναι αρκετά αναμενόμενα. Λόγω του μικρού αριθμού κρυφών επιπέδων, το μοντέλο δεν φαίνεται να μπορεί να κατηγοριοποιήσει καλά τα δεδομένα, αν και είναι τόσο λίγα. Μέχρι και την πέμπτη εποχή υπήρχε κάποια βελτίωση αλλά όχι σε τόσο μεγάλο βαθμό όσο χρειάζεται.



Εικόνα 4-14: Αποτελέσματα πρώτου δικτύου

Επίσης φαίνεται πως τα δεδομένα επαλήθευσης, δεν ακολουθούν τόσο κοντά τα δεδομένα εκπαίδευσης από άποψη απόδοσης, πράγμα που μπορεί να οφείλεται στο ότι επιλέχθηκαν αρκετά διαφορετικές εικόνες μέσα από τα τυχαία σετ που φτιαχτήκανε πιο πάνω.

4.4.3 Συμπεράσματα

Βλέποντας τα αποτελέσματα του δικτύου, υπάρχουν δύο διαφορετικά πορίσματα που βγαίνουν. Πρέπει να αυξηθεί η ακρίβεια του μοντέλου, μέσω περισσότερων κρυφών επιπέδων, αλλά και να ξεκινήσει η καταπολέμηση του overfitting.

4.5 Δεύτερο δίκτυο

Λαμβάνοντας υπόψη τα αποτελέσματα της πρώτης απόπειρας, πρέπει σίγουρα να προστεθούν κρυφά επίπεδα μέσα στο δίκτυο για να βελτιωθεί η απόδοση του, τουλάχιστον στα δεδομένα εκπαίδευσης. Αλλά προς το παρών δεν θα ασχοληθούμε με το overfitting, μέχρι να αυξηθεί αρκετά ακρίβεια.

4.5.1 Δημιουργία δεύτερου δικτύου με Hyperas.

Σε αυτό το στάδιο γίνεται χρήση του Hyperas, και θα παρουσιαστεί ο τρόπος με τον οποίο πρέπει να χτιστεί το δίκτυο.

Αρχικά για τη χρήση του Hyperas, πρέπει να δημιουργηθούν 2 διαφορετικές μέθοδοι. Η πρώτη είναι η data όπου πρέπει να προετοιμαστούν τα δεδομένα που θα πάρουν όλα τα δίκτυα που θα δημιουργηθούν. Όπως φαίνεται και από τον κώδικα, εδώ δεν υπάρχει διαφορά του τρόπου που προετοιμάζονται τα δεδομένα, σε σχέση με ένα κανονικό δίκτυο απλά πρέπει όλες οι διαδικασίες να γίνουν μέσα στην ίδια μέθοδο.

```
def data():
    dimension = 128
    batch_size = 32
    train_dir = './train/'
    validation_dir = './test/'
    train_datagen = ImageDataGenerator(rescale=1. / 255)

    validation_datagen = ImageDataGenerator(rescale=1.0 / 255.)

    train_generator = train_datagen.flow_from_directory(train_dir,
                                                        batch_size=batch_size,
                                                        class_mode='categorical',
                                                        target_size=(dimension, dimension))

    validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                                  batch_size=batch_size,
                                                                  class_mode='categorical',
                                                                  target_size=(dimension, dimension))

    return train_generator, validation_generator
```

Εικόνα 4-15: Προετοιμασία δεδομένων Hyperas

Η επόμενη και πιο σημαντική μέθοδος είναι η create_model. Εδώ φτιάχνεται το δίκτυο, με επιλογές. Όπως φαίνεται και από την ακόλουθη εικόνα, υπάρχει μόνο μία επιλογή για τα δύο convolutional επίπεδα, όσον αφορά το padding μέσω του padding_choice.

Το πρώτο convolutional επίπεδο, έχει επιλογή για τον αριθμό των φίλτρων, αλλά και του μεγέθους τους. Επίσης προστέθηκε και MaxPooling επίπεδα με διαστάσεις 2 επί 2, ελπίζοντας ότι μειώνοντας τις διαστάσεις των εικόνων επιτυεχθεί γρηγορότερη εκπαίδευση αυξάνοντας την ακρίβεια.

Στο δεύτερο επίπεδο, γίνεται μόνο επιλογή αριθμού φίλτρων, με σταθερή τη διάστασή τους, 3 επί 3. Η επιλογή μεγαλύτερων φίλτρων μετά το πρώτο επίπεδο δεν είναι συχνή και εφόσον χρησιμοποιούνται MaxPooling επίπεδα δεν υπάρχει λόγος να μειωθούν άλλο τις διαστάσεις.

Χρησιμοποιήθηκε και ένα σετ επιλογών και στο πλήρως συνδεδεμένο επίπεδο, ελπίζοντας να αυξηθεί η ακρίβεια του μοντέλου και από εκεί. Έπειτα φορτώνονται τα δεδομένα μέσω της fit_generator, και επιλέγεται πάλι σαν αριθμός των εποχών το 10.

```
def create_model():
    classes = 12
    dimension = 128
    batch_size = 32

    padding_choice = {{choice(['same', 'valid'])}}
    model = Sequential()

    model.add(Conv2D(filters={{choice([16, 32, 64, 128])}}, kernel_size={{choice([3, 5, 7])}}, padding=padding_choice,
                     activation='relu', input_shape=(dimension, dimension, 3)))
    model.add(MaxPooling2D(2, 2))
    model.add(BatchNormalization())

    model.add(Conv2D(filters={{choice([32, 64, 128])}}, kernel_size=3, padding=padding_choice,
                     activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dense({{choice([128, 256, 512])}}, activation='relu'))

    model.add(Dense(classes, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

    model.fit_generator(train_generator,
                        steps_per_epoch=train_generator.n // batch_size,
                        epochs=10,
                        validation_data=validation_generator,
                        validation_steps=validation_generator.n // batch_size)

    loss, acc = model.evaluate_generator(generator=validation_generator,
                                         steps=validation_generator.n // batch_size)
    print(model.summary())
    return {'loss': loss, 'status': STATUS_OK, 'model': model}
```

Εικόνα 4-16: Δημιουργία δικτύου Hyperas

Η μεταβλητή loss, που γυρνάει η μέθοδος evaluate, είναι το loss από τα δεδομένα επαλήθευσης, άρα η μόνη μετρική που πρέπει να ελαχιστοποιηθεί.

Έπειτα από κάθε «τρέξιμο» ενός δικτύου, γίνεται εκτύπωση των στοιχείων του, μέσω της model.summary(), οπότε ακόμα και αν το πρόγραμμα ξεμείνει από μνήμη να φανεί ποιο από τα μοντέλα που κατάφεραν να τρέξουν είχε την καλύτερη απόδοση και να χρησιμοποιηθεί ως βάση για την εκδοχή του.

Το επόμενο κομμάτι κώδικα, που χρειάζεται το Hyperas, είναι η μέθοδος ελαχιστοποίησης ανάλογα με τις επιλογές που έχουν τεθεί.

```
if __name__ == '__main__':
    train_generator, validation_generator = data()

    best_run, best_model = optim.minimize(model=create_model,
                                          data=data,
                                          algo=tpe.suggest,
                                          max_evals=10,
                                          trials=Trials(),
                                          eval_space=True)

    print('Best run:', best_run)
    print('Best model:', best_model.summary())
```

Εικόνα 4-17: Επιστροφή καλύτερου μοντέλου

Ο τρόπος δημιουργίας μοντέλων, αλλά και η αρχικοποίηση των δεδομένων, έχει ήδη περιγραφεί. Οι υπόλοιπες επιλογές που εμφανίζονται είναι το algo που πρέπει να γίνει επιλογή του αλγορίθμου που θα χρησιμοποιηθεί για την επιλογή των παραμέτρων στο create_model.

Εδώ, έχει γίνει επιλογή του προτεινόμενου tpe.suggest, (Tree-structured Parzen Estimator). Σε αντίθεση με πιο απλές προσεγγίσεις στο πρόβλημα της επιλογής των υπερ-παραμέτρων του δικτύου, όπως το grid search όπου επιλέγονται διαδοχικά όλοι οι πιθανοί συνδυασμοί από τις επιλογές που έχουμε δώσει, ή το random search που όπως φαίνεται από το όνομά του είναι η τυχαία επιλογή των παραμέτρων, μέσω του TPE επιλέγονται διαδοχικά παράμετροι που είναι πιθανότερο να οδηγήσουν σε δίκτυο με καλύτερη απόδοση.

Η παράμετρος max_evals χρησιμοποιείται για να επιλεγεί ο αριθμός των διαφορετικών δικτύων που θα τρέξει το Hyperas, εδώ 10, η παράμετρος trials είναι ένα αντικείμενο που χρησιμοποιεί το hyperopt για να αποθηκεύει τα ενδιάμεσα αποτελέσματα όλων των δικτύων και τέλος το eval_space χρησιμοποιείται ώστε στο τέλος να εκτυπωθούν οι ονομαστικές τιμές των επιλογών μας, και όχι η θέση τους. Σε περίπτωση δηλαδή που το padding που θα χρησιμοποιηθεί θα είναι το same, θα επιστραφεί η τιμή same αντί για 0.

4.5.2 Αποτελέσματα Hyperas

Από τις τελικές εκτυπώσεις στην κονσόλα, φαίνεται ποιο μοντέλο επέλεξε ως καλύτερο το Hyperas, το συνολικό χρόνο που πήρε η εκπαίδευση όλων των μοντέλων, το loss που κατάφερε το καλύτερο μοντέλο και φυσικά οι παράμετροι που το πέτυχαν.

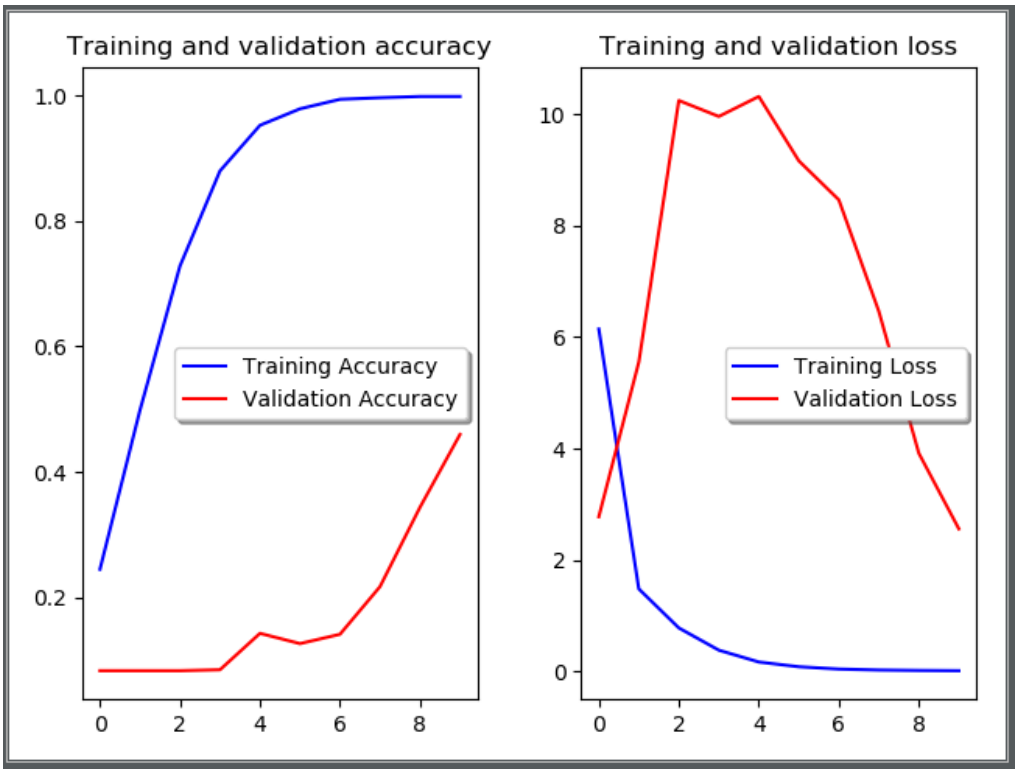
Στην συγκεκριμένη περίπτωση επιλέχθηκε zero-padding (same), 64 φίλτρα μεγέθους 5 επί 5 για το πρώτο επίπεδο και 32 φίλτρα για το δεύτερο convolutional επίπεδο, και 256 νευρώνες για το πλήρως συνδεδεμένο επίπεδο.

Παρατηρείται επίσης πως, με τη χρήση των έξτρα επιπέδων ο αριθμός των παραμέτρων που χρειάζονται στην εκπαίδευση έπεσε δραματικά. Από τα 65 εκατομμύρια πλέον χρειάζονται, σχεδόν, 8,5 εκατομμύρια, αλλά παρόλο που έπεσε ο αριθμός των παραμέτρων, το loss των δεδομένων επαλήθευσης μειώθηκε επίσης δραματικά. Το πρώτο μοντέλο είχε σχεδόν 13 loss σε αυτά τα δεδομένα ενώ τώρα «έπεσε» στο 2.5.

```
100%|                                        | 10/10 [31:30<00:00, 189.15s/it, best loss: 2.4412150382995605]
Best run: {'Dense': 256, 'filters': 64, 'filters_1': 32, 'kernel_size': 5, 'padding_choice': 'same'}
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 64)      4864
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 64)        0
_________________________________________________________________
batch_normalization_v1 (Batc (None, 64, 64, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_v1_1 (Ba (None, 32, 32, 32)        128
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 256)               8388864
_________________________________________________________________
dense_1 (Dense)              (None, 12)                3084
=================================================================
Total params: 8,415,660
Trainable params: 8,415,468
Non-trainable params: 192
_________________________________________________________________
```

Εικόνα 4-18: Αποτελέσματα Hyperas

Τρέχοντας, το δίκτυο με τις επιλογές του Hyperas, μπορούμε να δούμε από «κοντά» την διαδικασία εκπαίδευσης και να οπτικοποιήσουμε τα αποτελέσματα ώστε να βγουν τα καινούρια πορίσματα για το πως μπορεί να βελτιωθεί.



Εικόνα 4-19: Οπτικοποίηση αποτελεσμάτων δεύτερου δικτύου

Όπως φαίνεται, αυξήθηκε κατακόρυφα η ακρίβεια για τα δεδομένα εκπαίδευσης, φτάνοντας την ακρίβεια σε αυτά στο 100% μέσα σε 6 εποχές. Από την άλλη πλευρά το κενό μεταξύ της ακρίβειά τους και της ακρίβειας των δεδομένων επαλήθευσης μεγάλωσε πάρα πολύ, όπως ήταν αναμενόμενο.

4.6 Δημιουργία καινούριων δεδομένων

Για να καταπολεμηθεί το overfitting, υπάρχουν δύο διαφορετικές προσεγγίσεις. Ο πρώτος τρόπος είναι μέσω της τεχνητής δημιουργίας νέων εικόνων εκπαίδευσης είτε μέσω της χρήση των επιπέδων dropout.

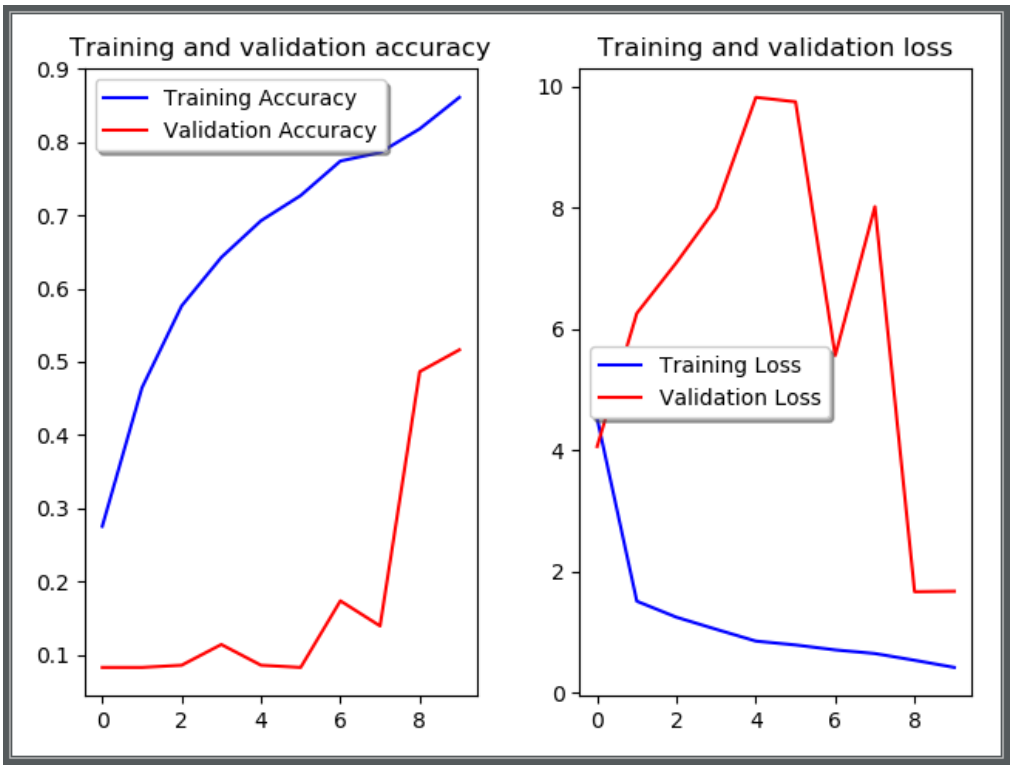
Αρχικά, επιλέγεται η δημιουργία των νέων εικόνων, ώστε με αυτό το τρόπο να μπορεί το δίκτυο να πετυχαίνει καλύτερα αποτελέσματα με εικόνες που, ενδεχομένως, δεν θα είναι «τραβηγμένες» με τον ίδιο τρόπο όπως αυτές που χρησιμοποιούνται στα ήδη υπάρχοντα δεδομένα εκπαίδευσης.

Για να επιτευχθεί αυτό το αποτέλεσμα, θα χρησιμοποιηθούν οι έξτρα παραμέτροι της κλάσης ImageDataGenerator του keras, χωρίς κάποια άλλες αλλαγές στην αρχιτεκτονική του νευρωνικού δικτύου.

```
def create_datagens():
    train_datagen = ImageDataGenerator(rescale=1.0 / 255.,
                                       horizontal_flip=True,
                                       vertical_flip=True)

    validation_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    return train_datagen, validation_datagen
```

Εικόνα 4-20: Τεχνητή αναστροφή εικόνων

Οι παράμετροι που χρησιμοποιούνται αρχικά είναι οι horizontal_flip και vertical_flip με τιμή true. Όπως είναι προφανές με αυτόν τον τρόπο, το keras μπορεί να αναστρέφει τυχαία εικόνες από τα δεδομένα εκπαίδευσης σε κάθε εποχή ώστε να γίνει πιο δύσκολη η εκπαίδευση του δικτύου με τεχνητά, «νέα», παραδείγματα.



Εικόνα 4-21: Αποτελέσματα με αναστροφή εικόνων

Όπως φαίνεται το δίκτυο δυσκολεύτηκε παραπάνω στην εκπαίδευση, τόσο που δεν μπόρεσε να πιάσει το 100% στην ακρίβεια μέχρι και τη δέκατη εποχή. Στις τελευταίες εποχές φαίνεται πως μειώθηκε λίγο περισσότερο το loss για τα δεδομένα επαλήθευσης, αλλά όχι τόσο όσο χρειάζεται, καθώς υπάρχει ακόμα πολύ overfitting στο δίκτυό.

Για αυτό το λόγο, θα χρησιμοποιηθούν ακόμα περισσότερες παράμετροι για τη δημιουργία νέων εικόνων μέσω του ImageDataGenerator. Παρατηρώντας τις εικόνες των δεδομένων εκπαίδευσής, φαίνεται πως κάποιες από τις εικόνες έχουν διαφορετικά επίπεδα μεγέθυνσης αλλά και διαφέρουν σε κάθε φυτό, η κατεύθυνση στην οποία «πηγαίνουν» τα φύλλα.

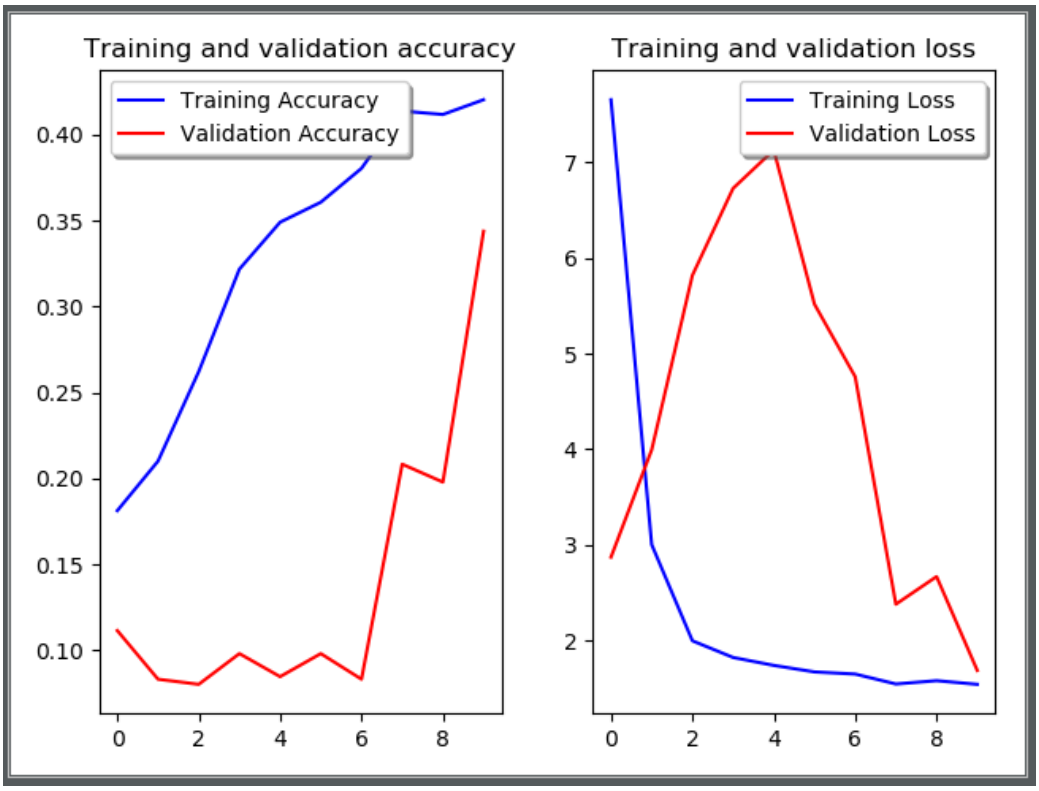
```
def create_datagens():
    train_datagen = ImageDataGenerator(rescale=1.0 / 255.,
                                       rotation_range=30,
                                       width_shift_range=0.3,
                                       height_shift_range=0.3,
                                       zoom_range=0.3,
                                       fill_mode='nearest',
                                       horizontal_flip=True,
                                       vertical_flip=True)

    validation_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    return train_datagen, validation_datagen
```

Εικόνα 4-22: Μεγαλύτερη παραμόρφωση των δεδομένων εκπαίδευσης

Για να αντικατοπτριστούν αυτές τις διαφορές μεταξύ των εικόνων, στα δεδομένα εκπαίδευσης, προστίθενται οι παρακάτω παράμετροι: rotation_range, μέσω της οποίας το keras περιστρέφει τις εικόνες από 0 έως 30 μοίρες (χωρίς τις υποδιαιρέσεις τους), το width_shift_range και height_shift_range, μέσω των οποίων «μετακινείται» μια εικόνα κατά ένα ποσοστό προς τα αριστερά / δεξιά ή προς τα πάνω / κάτω.

Το zoom_range είναι το ποσοστό μεγέθυνσης που θα χρησιμοποιηθεί, ενώ το fill_mode είναι ο τρόπος με τον οποίο θα υπολογιστούν οι τιμές των καινούριων εικονοστοιχείων των εικόνων μετά από τις παραμορφώσεις όπως το rotation, width/height shit. Με το nearest, θα επιλεχθεί η τιμή του κοντινότερου εικονοστοιχείου για να υπολογιστεί η τιμή του καινούριου.

Τρέχοντας πάλι το ίδιο δίκτυο, με αυτά τα «καινούρια» δεδομένα, τα διαγράμματα ακρίβειας και loss εμφανίζονται ως κάτωθι :



Εικόνα 4-23: Απόδοση δικτύου σε δεδομένα με πολλαπλές παραμορφώσεις

Όπως ήταν αναμενόμενο, λόγω των «καινούριων» δεδομένων η ακρίβεια των αποτελεσμάτων, έπεσε ακόμα περισσότερο, αλλά φαίνεται ότι στις τελευταίες εποχές το lost των δεδομένων επαλήθευσης έχει φτάσει πολύ κοντά, αυτό των δεδομένων εκπαίδευσης.

4.7 Δημιουργία τρίτου δικτύου

Μετά από όλες αυτές τις αλλαγές, πρέπει να χρησιμοποιηθεί πάλι το Hyperas, ώστε να γίνει προσπάθεια αύξησης, πάλι, της ακρίβειας του μοντέλου, αλλά και να κρατηθεί σε χαμηλό επίπεδο το overfitting.

Για να επιτευχθεί ο πρώτος στόχος, πρέπει να προστεθεί τουλάχιστον ένα ακόμα convolutional επίπεδο, ενώ για τον δεύτερο ένα dropout επίπεδο μετά από το πλήρως συνδεδεμένο.

Με παρόμοιες επιλογές όπως και πριν, ο κώδικας του Hyperas γίνεται ως εξής:

```
def create_model():
    classes = 12
    dimension = 128
    batch_size = 32

    model = Sequential()

    model.add(Conv2D(filters={{choice([32, 64, 128])}}, kernel_size={{choice([3, 5, 7])}}, padding='same',
                     activation='relu', input_shape=(dimension, dimension, 3)))
    model.add(MaxPooling2D(2, 2))
    model.add(BatchNormalization())

    model.add(Conv2D(filters={{choice([32, 64, 128])}}, kernel_size=3, padding='same',
                     activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(BatchNormalization())

    model.add(Conv2D(filters={{choice([32, 64, 128])}}, kernel_size=3, padding='same',
                     activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dense({{choice([128, 256, 512, 1024])}}, activation='relu'))
    model.add(Dropout({{{uniform(0, 1)}}}))

    model.add(Dense(classes, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

    model.fit_generator(train_generator,
                        steps_per_epoch=train_generator.n // batch_size,
                        epochs=10,
                        validation_data=validation_generator,
                        validation_steps=validation_generator.n // batch_size)

    loss, acc = model.evaluate_generator(generator=validation_generator,
                                         steps=validation_generator.n // batch_size)
    print(model.summary())
    return {'loss': loss, 'status': STATUS_OK, 'model': model}
```

Εικόνα 4-24: Hyperas δίκτυο με 3 convolutional επίπεδα και dropout

Η επιλογή του dropout είναι ελαφρώς διαφορετική από τις προηγούμενες, καθώς δεν δηλώνονται διακριτές τιμές που μπορεί να πάρει από το επίπεδο, αλλά όλες οι υποδιαιρέσεις μεταξύ του εύρους 0 έως και 1.

Τρέχοντας μερικά δίκτυα, με αυτές τις επιλογές, το καλύτερο αποτέλεσμα που επέστρεψε το Hyperas είναι το εξής: 128 φίλτρα μεγέθους 7 επί 7 στο πρώτο επίπεδο, 64 φίλτρα στο δεύτερο και τρίτο επίπεδο και 256 νευρώνες στο πλήρως συνδεδεμένο με dropout της τάξης του 20%.

```
100%|                    | 15/15 [1:01:21<00:00, 246.48s/it, best loss: 1.759821579569862]
Best run: {'Dense': 256, 'Dense_1': 0.20670036025517124, 'filters': 128, 'filters_1': 64, 'filters_2': 64, 'kernel_size': 7}
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 128, 128, 128)     18944
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 64, 64, 128)       0
_________________________________________________________________
batch_normalization_v1_21 (B (None, 64, 64, 128)       512
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 64, 64, 64)        73792
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_v1_22 (B (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_v1_23 (B (None, 16, 16, 64)        256
_________________________________________________________________
flatten_7 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_14 (Dense)             (None, 256)               4194560
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_15 (Dense)             (None, 12)                3084
=================================================================
Total params: 4,328,332
Trainable params: 4,327,820
Non-trainable params: 512
_________________________________________________________________
```

Εικόνα 4-25: Παραγμένο δίκτυο από το Hyperas

Το δίκτυο αυτό δεν είναι το ιδεατό, καθώς κοιτώντας τα στοιχεία μετά το τρέξιμο της τελευταίας εποχής, η ακρίβεια των δεδομένων εκπαίδευσης είναι στο 41% και των δεδομένων επαλήθευσης στο 34%. Πρέπει σίγουρα να αυξηθεί η ακρίβεια του μοντέλου αλλά τουλάχιστον πλέον το overfitting δεν είναι τόσο μεγάλο πρόβλημα.

Εφόσον έπεσε τόσο πολύ η ακρίβεια, θα χρειαστεί να αυξηθεί η χρονική περίοδος που θα εκπαιδεύεται το νευρωνικό δίκτυο. Οπότε, οι εποχές εκπαίδευσης πλέον θα είναι 100, αλλά ταυτόχρονα, είναι καλό, εφόσον θα πάρει πολύ παραπάνω ώρα η όλη διαδικασία, να ελέγχεται μετά από κάθε εποχή η «ποιότητα» του δικτύου και να αποθηκεύονται οι τιμές των παραμέτρων του.

Για να επιτευχθεί αυτό, γίνεται χρήση των callbacks. Τα callbacks είναι μέθοδοι που τρέχουν σε διάφορα στάδια της εκπαίδευσης. Σε αυτή την περίπτωση θα χρησιμοποιηθεί ένα από τα έτοιμα callbacks του keras, το ModelCheckpoint.

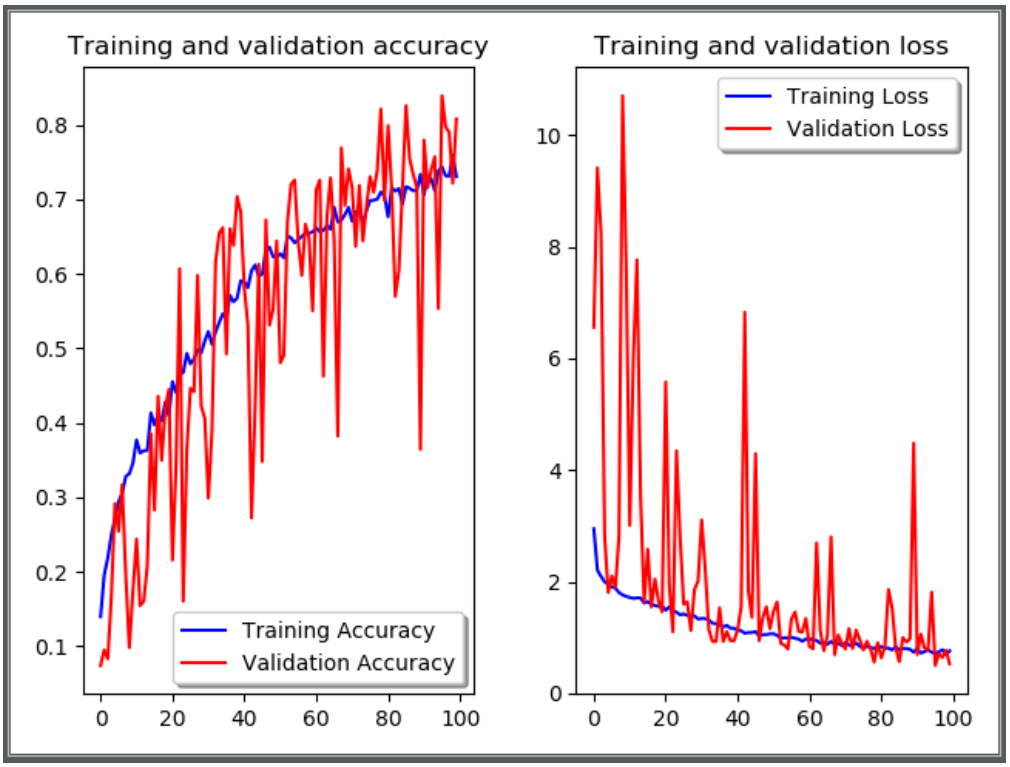
Σε αυτό το callback, δηλώνεται η τιμή που θα παρακολουθεί, και σε ποια μεταβολή της να αποθηκεύονται τα βάρη των παραμέτρων. Εδώ, βάζοντας για παρακολούθηση την ακρίβεια των δεδομένων επαλήθευσης (val_acc), μας ενδιαφέρει πότε θα αυξηθεί η τιμή της, άρα το mode είναι max. Με το save_best_only επιλέγεται αν θα αποθηκεύεται μόνο το καλύτερο μοντέλο, ενώ το verbose χρησιμοποιείται για το αν θα εκτυπώνονται πληροφορίες στην κονσόλα ή όχι.

```
filepath = "weights-improvement-3layer-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=False, mode='max')
callbacks_list = [checkpoint]

history = model.fit_generator(train_generator,
                              epochs=epochs,
                              validation_data=validation_generator,
                              callbacks=callbacks_list)
```

Εικόνα 4-26: Παρακολούθηση του μοντέλου με callbacks

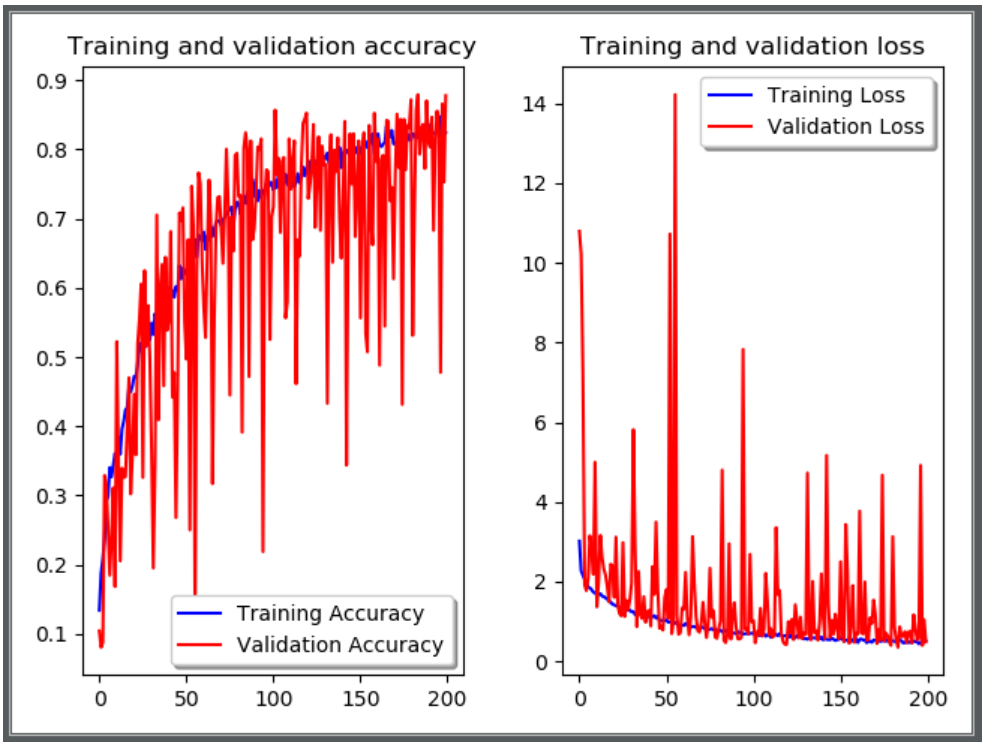
4.7.1 Αποτελέσματα τρίτου δικτύου

Τρέχοντας αυτό το δίκτυο, παρόλο υπάρχουν τυχαίες μεταβολές του loss για τα δεδομένα επαλήθευσης, φαίνεται ότι ακολουθεί το loss των δεδομένων εκπαίδευσης αρκετά κοντά, άρα δεν έχουμε πρόβλημα overfitting.



Εικόνα 4-27: Αποτελέσματα τέταρτου δικτύου

Τρέχοντας το ίδιο ακριβώς δίκτυο, αυτή τη φορά για 200 εποχές φαίνεται πως το μοντέλο συνεχίζει να αυξάνει την ακρίβεια, αλλά με πολύ χαμηλούς ρυθμούς. Καθώς ένας τόσο μεγάλος αριθμός εποχών είναι υπερ-αρκετός για ένα τόσο μικρό dataset, μπορεί να γίνει περαιτέρω διεύρυνση του δικτύου.



Εικόνα 4-28: Εκπαίδευση δικτύου με 3 επίπεδα convolutions για 200 εποχές

4.8 Δημιουργία τέταρτου δικτύου

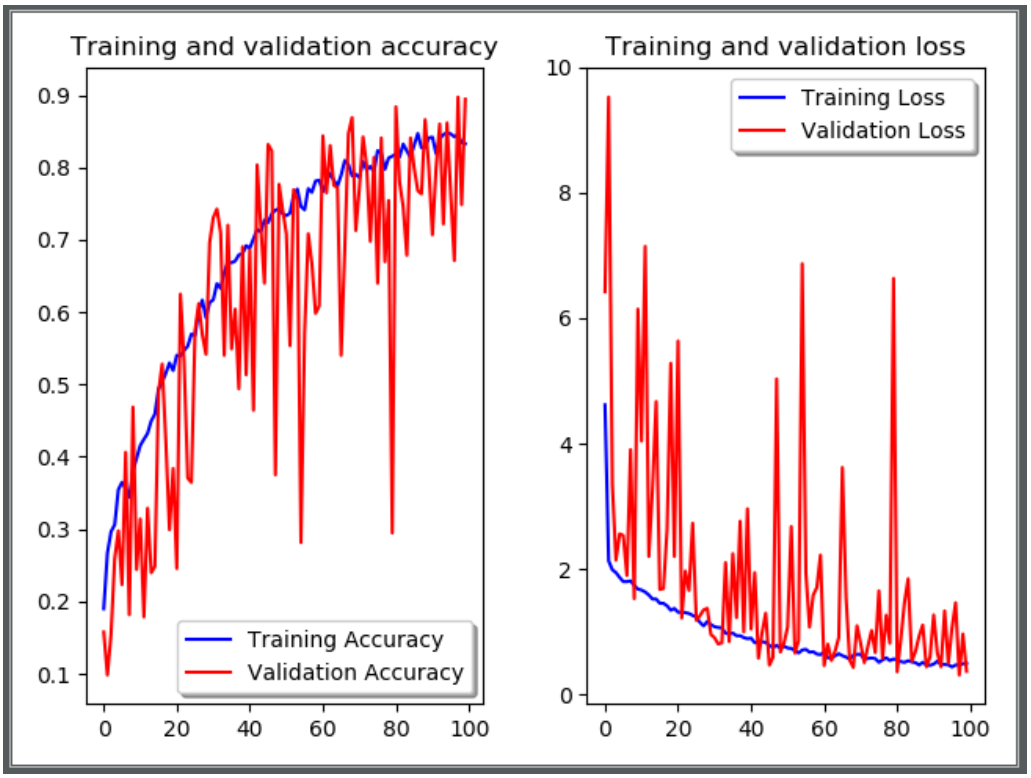
Με βάση τα αποτελέσματα από την εκτέλεση του προηγούμενου δικτύου, φαίνεται πως μπορεί να γίνει προσθήκη παραπάνω επιπέδων για να αυξηθεί η ακρίβεια των αποτελεσμάτων. Για να μην «σπαταληθεί» πολύ χρόνος, τρέχοντας πολλές διαφορετικές επιλογές στο Hyperas, αυτή τη φορά θα δημιουργηθεί πρώτα ένα μοντέλο με 4 convolutional επίπεδα, ώστε να φανεί το δίκτυο θα πετύχει καλά αποτελέσματα ή αν θα χρειαστεί περισσότερα, και μετά μέσω του Hyperas μπορούν να γίνουν οι οποιεσδήποτε μικρο-βελτιώσεις.

```
model = Sequential()
model.add(Conv2D(64, 5, padding='same', activation='relu', input_shape=(dimension, dimension, 3)))
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Conv2D(128, 3, padding='same', activation='relu'))
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Conv2D(128, 3, padding='same', activation='relu'))
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Conv2D(128, 3, padding='same', activation='relu'))
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(classes, activation='softmax'))
```

Εικόνα 4-29: Δίκτυο με 4 convolutional επίπεδα

Καθότι δεν φαίνεται να προτιμώνται ιδιαίτερα μεγάλος αριθμός φίλτρων σε κάθε επίπεδο, επιλέχθηκαν 64 για το πρώτο επίπεδο, και 128 για όλα τα επόμενα, αλλά αυξήθηκε και ο αριθμός των νευρώνων στο τελευταίο επίπεδο στους 1024, ελπίζοντας ότι βοηθήσει και αυτό στην αύξηση της ακρίβειας, αλλά ταυτόχρονα αυξήθηκε και το ποσοστό Dropout στο 50% ώστε να αποφύγουμε το overfitting.

4.8.1 Αποτελέσματα τέταρτου δικτύου



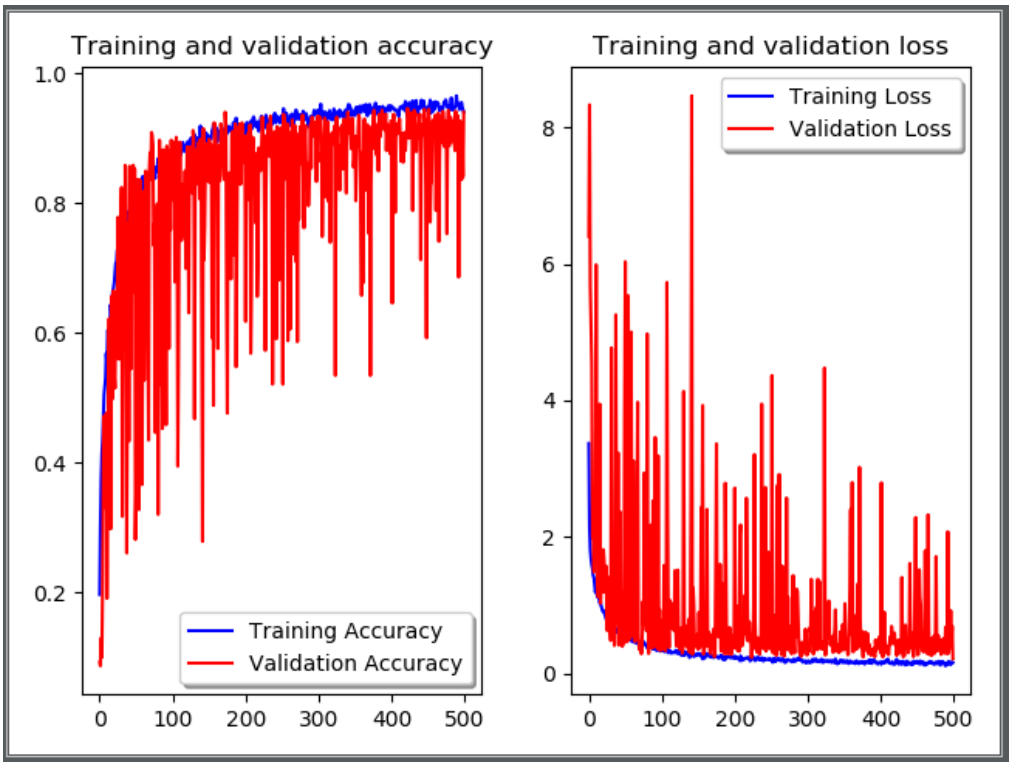
Εικόνα 4-30: Δίκτυο με 4 convolutional επίπεδα

Τρέχοντας και αυτό το δίκτυο για 100 εποχές φαίνεται πως βελτιώθηκαν τα αποτελέσματα του δικτύου σχεδόν κατά 10%, από 70% ακρίβεια στα δεδομένα εκπαίδευσης στο 80%.

Εφόσον δεν υπάρχουν σοβαρές ενδείξεις για overfitting, χρησιμοποιήθηκε το Hyperas, για να φανούν ποιες επιλογές θα αποδίδουν αρκετά καλά όσον αφορά τα loss των δεδομένων επαλήθευσης. Τα αποτελέσματα που επιστραφήκανε ήταν τα εξής: από το πρώτο έως το 4ο επίπεδο: 256, 128, 32, 32 φίλτρα για τα convolutional επίπεδα και dropout 43% έπειτα από το πλήρες συνδεδεμένο επίπεδο 1024 νευρώνων.

Για να επιτευχθεί, η μέγιστη απόδοση του δικτύου, χρησιμοποιήθηκαν οι επιλογές του Hyperas αλλά αυτή τη φορά το δίκτυο εκπαιδευότανε για 500 εποχές, και

αποθήκευε, όπως και πριν, τα βάρη των δικτύων που είχαν την μεγαλύτερη ακρίβεια τα δεδομένα επαλήθευσης.



Εικόνα 4-31: Εκπαίδευση δικτύου για 500 εποχές

Φαίνεται υπήρξε πολύ ελαφρύ overfitting αλλά, το καλύτερο αποτέλεσμα που καταγράφηκε, κατάφερε να πιάσει 95% ακρίβεια.

4.9 Δημιουργία αρχείου csv

Για να βαθμολογηθεί η ποιότητα του δικτύου στα δεδομένα του διαγωνισμού, θα χρειαστεί να «φορτωθούν» τα βάρη σε ένα δίκτυο, με ακριβώς την ίδια αρχιτεκτονική όπως αυτό που τα δημιούργησε. Αυτό μπορεί να γίνει εύκολα με τη χρήση της εντολής, load_weights, που παίρνει ως παράμετρο το όνομα του αρχείου.

Εφόσον γίνει αυτό, μπορούν να «φορτωθούν» τα αρχεία που πρέπει να προβλεφθεί η κλάση τους σε αυτό. Εδώ χρειάζεται προσοχή στο ότι τα αρχεία πρέπει να δοθούν με τις ίδιες διαστάσεις όπως και τα δεδομένα εκπαίδευσης, οπότε η χρήση ενός generator του keras, απλοποιεί αρκετά την κατάσταση.

```
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_files,
        target_size=(128, 128),
        color_mode="rgb",
        shuffle=False,
        class_mode='categorical',
        batch_size=32)
```

Εικόνα 4-32: Φόρτωμα αρχείων για testing

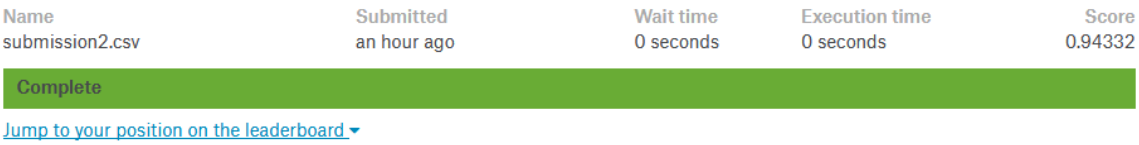
Στο test_files είναι δηλωμένος ο φάκελος που εμπεριέχονται τα αρχεία, στο target_size επιλέγονται οι διαστάσεις που πρέπει να έχουν οι εικόνες, η σειρά με την οποία είναι αποθηκευμένα τα χρώματα στις εικόνες, με το shuffle μπορούν να φορτωθούν με τυχαία σειρά οι εικόνες, που στην προκειμένη περίπτωση δεν έχει κάποια σημασία, όπως επίσης και το class_mode καθώς δεν έχουμε τα δεδομένα σε πολλαπλούς φακέλους, ανάλογα με τη κλάση τους. Τέλος με το batch_size φορτώνονται λίγες, λίγες τις εικόνες ώστε να μην γεμίσει η RAM.

```
filenames = test_generator.filenames
nb_samples = len(filenames)

predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/32))

categories = []

for i in range(predict.argmax(axis=1).size):
    categories.append(class_names[predict.argmax(axis=1)[i]])

sub_file = pd.DataFrame(data={'file': names, 'species': categories})
sub_file.to_csv('submission.csv', index=False)
```

Εικόνα 4-33: Πρόβλεψη κατηγοριών και αποθήκευση σε αρχείο csv

Έπειτα, εφόσον γίνουν οι προβλέψεις φορτώνοντας όλα τα δεδομένα, δημιουργούνται οι πίνακες που εμπεριέχουν τα ονόματα των αρχείων αλλά και την ονομασία της κατηγορίας στην οποία το δίκτυο πιστεύει ότι ανήκουν. Με τη χρήση του πακέτου pandas η δημιουργία ενός αρχείου csv μπορεί να γίνει εύκολα και γρήγορα, και θα εμπεριέχει τα αποτελέσματα σε μορφή που την δέχεται ο διαγωνισμός.

Ανεβάζοντας το αρχείο που παράχθηκε στη σελίδα του διαγωνισμού το αποτέλεσμα ήταν 0,94332 (με άριστα το 1), αλλά δυστυχώς λόγω του ότι ο διαγωνισμός είχε λήξει στο σημείο που έγινε προσπάθεια ανεβάσματος, το αποτέλεσμα δεν φαίνεται στον τελικό πίνακα κατάταξης.



Εικόνα 4-34: Πρώτο αποτέλεσμα Kaggle

4.10 Προσπάθειες μικρό αυξήσεων ακρίβειας

Εφόσον έχει ολοκληρωθεί το κυρίως δίκτυο, μπορούν να γίνουν κάποιες μικρές αλλαγές, όπως η χρήση διαφορετικών optimizers και activation functions, για να φανεί αν ένας διαφορετικός συνδυασμός, ίσως καταφέρει να αυξήσει κατά ένα μικρό ποσοστό την ακρίβεια του μοντέλου.

Το keras προσφέρει διάφορους optimizers, έτοιμους για χρήση, όπως τον Adam που χρησιμοποιήθηκε καθ' όλη την διάρκεια των δοκιμών. Ένας άλλος είναι ο RMSProp, όπου πέρα από το learning rate, είναι προτεινόμενο να αφήσουμε τις λοιπές παραμέτρους ως έχουν.

Μία άλλη κλασική επιλογή είναι ο stochastic gradient descent, στον οποίο μπορεί να γίνει επιλογή του learning rate, τον ρυθμό με τον οποίο θα πέφτει (decay), την «ορμή», momentum, καθώς και αν θα γίνεται χρήση του Nesterov momentum.

```
opt = SGD(lr=1, decay=1/500, momentum=0.9, nesterov=True)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])
```

Εικόνα 4-35: Επιλογή SGD για optimizer

Το ίδιο πράγμα μπορεί να γίνει και στα activation επίπεδα του δικτύου. Παρόλο που η χρήση των relu θεωρείται πλέον η πιο συχνή, μπορεί να γίνει δοκιμή και άλλων απλών activation function όπως η tanh, που μπορούν να χρησιμοποιηθούν ακριβώς όπως και η relu, με απλή αντικατάσταση μέσα στις παραμέτρους ενός convolutional επιπέδου, είτε με πιο «προχωρημένες» activation functions όπως η LeakyReLu, όπου θα χρειαστεί η προσθήκη ενός διαφορετικού επίπεδου στο δίκτυο.

```
model.add(Conv2D(64, 5, padding='same', input_shape=(dimension, dimension, 3)))
model.add(LeakyReLU())
```

Εικόνα 4-36: Προσθήκη LeakyReLu activation

Δυστυχώς κατά τη διάρκεια των δοκιμών, με διαφορετικές activation functions ή optimizers δεν επιτεύχθηκε κάποια αύξηση της ακρίβειας του μοντέλου.

Αλλά αλλάζοντας το batch size (που θεωρείται και αυτό ως μία υπερ-παράμετρος), αυξήθηκε η ακρίβεια στα δεδομένα testing στο 0,95088.

Σαν τελευταία προσπάθεια, έχοντας φορτώσει τα βάρη στο ίδιο μοντέλο χρησιμοποιήθηκαν διαφορετικά δεδομένα για την εκπαίδευση. Αυτή τη φορά αντί να έχουν όλες οι κατηγορίες τον ίδιο αριθμό δεδομένων, φορτώθηκαν το 75% όλων των δεδομένων για εκπαίδευση και το υπόλοιπο 25% έμειναν για δεδομένα επαλήθευσης, για 500 εποχές εκπαίδευσης.

Με αυτό τον τρόπο αυξήθηκε η ακρίβεια του μοντέλου στο 97%, τοπικά, αλλά και η ακρίβεια των δεδομένων αξιολόγησης του kaggle, στο 96.7%.

Εικόνα 4-37: Τελική ακρίβεια του μοντέλου

5 Επίλογος

5.1 Σύνοψη και συμπεράσματα

Πλέον η ενασχόληση με διαφορετικά πεδία της πληροφορικής χωρίς να υπάρχει πρότερη «επαφή» με αυτό είναι πιο εύκολη από ποτέ. Το πρόβλημα είναι να μπορέσουμε να ξεχωρίσουμε ποιες πηγές προσφέρουν την γνώση που χρειάζεται για το κομμάτι με το οποίο θέλουμε να ασχοληθούμε.

Στα πλαίσια της διπλωματικής αναφέρθηκα στα πιο κύρια σημεία, που πρέπει να γνωρίζει κάποιος που θέλει να ξεκινήσει να ασχολείται με τα νευρωνικά δίκτυα, από το μηδέν, χωρίς να μπω σε πολύωρες επεξηγήσεις που αφορούν το μαθηματικό κομμάτι της ενημέρωσης των βαρών των νευρώνων ή τον τρόπο με τον οποίο δουλεύει ξεχωριστά ο κάθε optimizer.

5.2 Όρια και περιορισμοί της έρευνας

Καθώς χρησιμοποιήθηκε ο προσωπικός μου υπολογιστής για την εκπαίδευση των δικτύων, χρειάστηκε να περιοριστώ σε dataset με σχετικά μικρό αριθμό δεδομένων ώστε να μπορέσω να εκπαιδεύσω τα δίκτυα, σε εύλογα χρονικά περιθώρια, κυρίως στις περιπτώσεις όπου ο χρόνος εκπαίδευσης ήταν στις 500 εποχές, όταν προσπαθούσα να τρέξω πολλαπλά δίκτυα μέσω του Hyperas, ο χρόνος που χρειαζόταν για την εκπαίδευσης 30 δικτύων ξεπερνούσε την μία μέρα συνεχόμενης χρήσης, και μπορούσε ανά πάσα στιγμή να γεμίσει η μνήμη.

5.3 Μελλοντικές Επεκτάσεις

Υπάρχουν και πολλά άλλα εργαλεία που βοηθούν στη γρήγορη δημιουργία νευρωνικών δικτύων που θεωρούνται ότι είναι ίσως και πιο απλά από το keras με βάση το TensorFlow. Τέτοια εργαλεία είναι το PyTorch και το Theano.

Επίσης μπορούν να παρουσιαστούν και διαφορετικές μεθοδολογίες για την επίτευξη ενός καλού αποτελέσματος σε έναν διαγωνισμό του Kaggle όπως είναι η χρήση ενός ήδη εκπαιδευμένου δικτύου όπως το Xception, VGG16, ResNet50 και άλλα που φαίνεται ότι έχουν πολύ καλή απόδοση με λίγη εκπαίδευση σε παρόμοια dataset με αυτά που εκπαιδεύτηκαν.

Βιβλιογραφία

Novet, J., 2017. Everyone keeps talking about A.I.—here's what it really is and why it's so hot now [WWW Document]. CNBC. URL https://www.cnbc.com/2017/06/17/what-is-artificial-intelligence.html (accessed 9.21.19).

Chollet, F., 2018. Deep learning with Python. Manning Publications Co, Shelter Island, New York.

Samuel, A.L., 1959. Some Studies in Machine Learning Using the Gameof Checkers. II-Recent Progress. MACHINE LEARNING 17.

Mitchell, T.M., 1997. Machine Learning, McGraw-Hill series in computer science. McGraw-Hill, New York.

McCulloch, W.S., Pitts, W., 1943. Neurocomputing: Foundations of Research, in: Anderson, J.A., Rosenfeld, E. (Eds.), . MIT Press, Cambridge, MA, USA, pp. 15–27.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408. https://doi.org/10.1037/h0042519

Minsky, M., Papert, S.A., 1969. Perceptrons: An Introduction to Computational Geometry. MIT Press.

Werbos, P.J., 1975. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1988. Neurocomputing: Foundations of Research, in: Anderson, J.A., Rosenfeld, E. (Eds.), . MIT Press, Cambridge, MA, USA, pp. 696–699.

LeCun, Y., Bottou, L., Orr, G.B., Müller, K.-R., 1998. Efficient BackProp, in: Neural Networks: Tricks of the Trade, This Book Is an Outgrowth of a 1996 NIPS Workshop. Springer-Verlag, London, UK, UK, pp. 9–50.

Herculano-Houzel, S., 2009. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. Front Hum Neurosci 3. https://doi.org/10.3389/neuro.09.031.2009

Rosebrock, A., 2017. Deep Learning for Computer Vision with Python: Starter Bundle. PyImageSearch.

Tang, A., Tam, R., Cadrin-Chênevert, A., Guest, W., Chong, J., Barfett, J., Chepelev, L., Cairns, R., Mitchell, J., Cicero, M., Gaudreau Poudrette, M., Jaremko, J., Md, C., Gallix, B., Gray, B., Geis, R., O'Connell, T., Babyn, P., Koff, D., Shabana, W., 2018. Canadian Association of Radiologists White Paper on Artificial Intelligence in Radiology. Canadian Association of Radiologists Journal 69. https://doi.org/10.1016/j.carj.2018.02.002

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs].

Cherry, E.C., 1953. Some experiments on the recognition of speech, with one and with two ears. Journal of the Acoustical Society of America 25, 975–979. https://doi.org/10.1121/1.1907229

Sra, S., Nowozin, S., Wright, S.J., 2012. Optimization for Machine Learning. MIT Press.

Kaggle: Your Home for Data Science [WWW Document], n.d. URL https://www.kaggle.com/ (accessed 5.26.19).

Plant Seedlings Classification [WWW Document], n.d. URL https://kaggle.com/c/plant-seedlings-classification (accessed 5.26.19).

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges [WWW Document], n.d. URL http://yann.lecun.com/exdb/mnist/ (accessed 5.26.19).

Fashion MNIST [WWW Document], n.d. URL https://kaggle.com/zalando-research/fashionmnist (accessed 5.26.19).

ImageNet [WWW Document], n.d. URL http://www.image-net.org/ (accessed 5.26.19).

Google Landmark Recognition 2019 [WWW Document], n.d. URL https://kaggle.com/c/landmark-recognition-2019 (accessed 5.26.19).

TensorFlow [WWW Document], n.d. URL https://www.tensorflow.org/about/case-studies (accessed 7.5.19).

TensorFlow in Anaconda, 2018. . Anaconda. URL https://www.anaconda.com/tensorflow-in-anaconda/ (accessed 9.21.19).