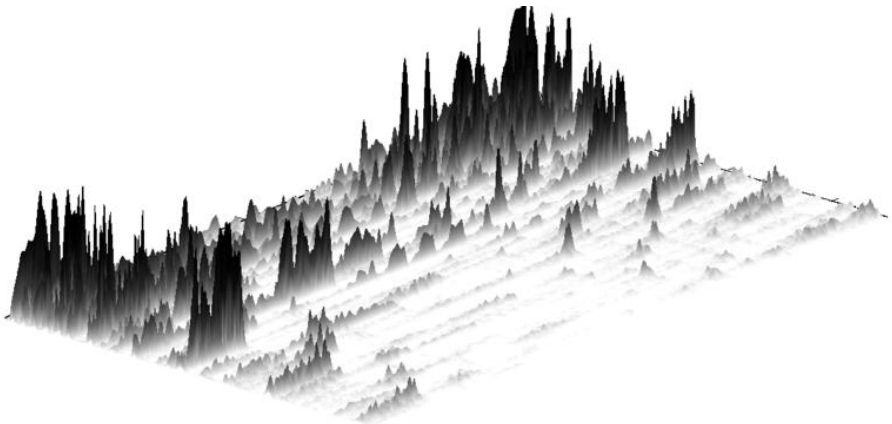
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

Π.Μ.Σ. «Προηγμένα Συστήματα Πληροφορικής - Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Αναγνώριση Ομιλίας και Ήχου



Δαβράδου Αγάπη

Πάνου Γιώργος

Σπυρόπουλος Κωνσταντίνος

Καθηγητής: Πικράκης Άγγελος

Αθήνα
30/11/2019

Περίληψη

Η παρούσα εργασία υλοποιήθηκε στα πλαίσια του μαθήματος «Αναγνώριση Ομιλίας και Ήχου» και αποσκοπεί στην υλοποίηση ενός Automatic Speech Recognition (ASR) συστήματος. Το τελικό σύστημα δέχεται ως είσοδο μία ηχογράφηση, η οποία συνιστά πρόταση αποτελούμενη από 4-10 ψηφία, από το 0 έως το 9, και τα οποία έχουν ειπωθεί με αρκούντως μεγάλα διαστήματα παύσης, περίπου ένα ψηφίο ανά δευτερόλεπτο. Το σύστημα έχει υλοποιηθεί έτσι, ώστε να μην εξαρτάται από τα χαρακτηριστικά της φωνής του ομιλητή. Στην έξοδό του το σύστημα, αφού κατατμήσει την πρόταση και αναγνωρίσει κάθε λέξη, παράγει σε μορφή κειμένου τα αποτελέσματα που αναγνωρίζει.

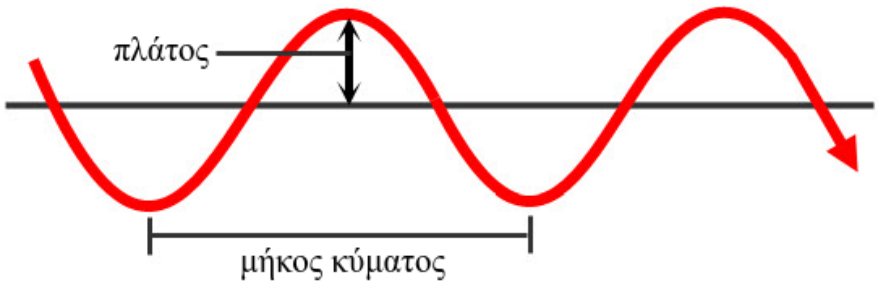
Περιεχόμενα

Κεφάλαιο 1:

Εισαγωγή

1.1 Ψηφιακή επεξεργασία σήματος

Το πρώτο βήμα της ψηφιακής επεξεργασίας σήματος είναι η μετατροπή ενός αναλογικού συστήματος σε ψηφιακό (δηλαδή η μετατροπή ενός συνεχούς σήματος σε διακριτό), κάνοντας χρήση ενός μετατροπέα αναλογικού σήματος σε ψηφιακό (analog-to-digital converter - ADC), ο οποίος μετασχηματίζει το αναλογικό σήμα σε μια ακολουθία από αριθμούς.

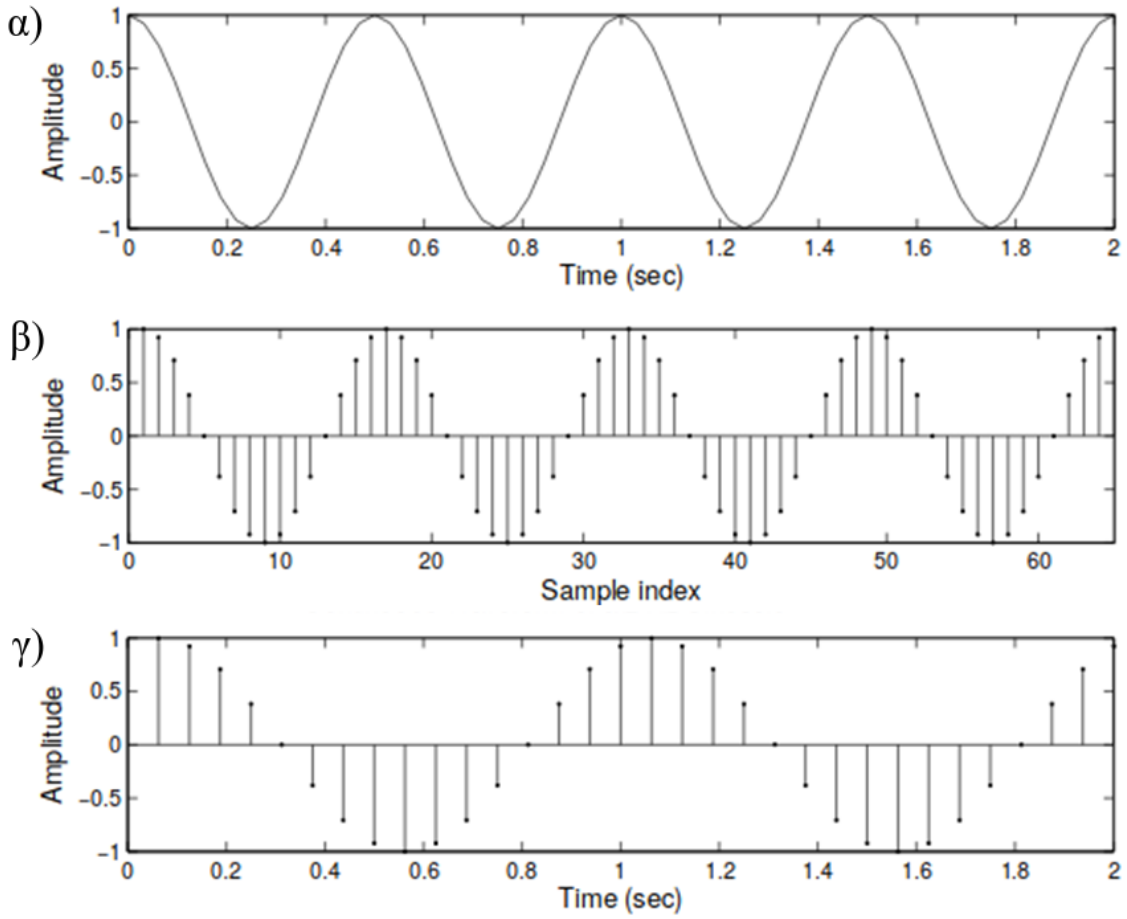


Σχήμα **1.1:** Παράμετροι ηχητικού σήματος.

Η μετατροπή πραγματοποιείται μέσω της δειγματοληψίας (sampling), η οποία συνήθως πραγματοποιείται σε δύο στάδια, στο στάδιο της διακριτοποίησης (discretization) και της κβάντισης σήματος (quantization). Η διακριτοποίηση σημαίνει ότι το σήμα χωρίζεται σε ίσα χρονικά διαστήματα και κάθε διάστημα αντιπροσωπεύεται από μία μόνο μέτρηση πλάτους (amplitude). Η κβάντιση σήματος σημαίνει ότι η κάθε μέτρηση πλάτους προσεγγίζεται με μία τιμή από ένα πεπερασμένο σύνολο.

Η συχνότητα δειγματοληψίας η ρυθμός δειγματοληψίας μετριέται σε Hz και εκφράζει το πλήθος δειγμάτων που λαμβάνονται σε διάρκεια ενός δευτερολέπτου. Για παράδειγμα,

εάν η συχνότητα f_s = 10.000Hz, σημαίνει ότι ο δειγματολήπτης λαμβάνει 10.000 δείγματα ανά δευτερόλεπτο σήματος. Η χρονική απόσταση των δειγμάτων υπολογίζεται από τον τύπο $f_s = \frac{1}{T}$, όπου T η περίοδος της δειγματοληψίας. Αντίστοιχα, για μία συχνότητα δειγματοληψίας f_s = 10.000Hz, η περίοδος θα είναι T = $\frac{1}{10,000}$ = 0.0001 δευτερόλεπτα.



Σχήμα **1.2:** Αναπαράσταση μίας συνεχής κυματομορφής ημιτονοειδούς κύματος (α), του αντίστοιχου δειγματοληπτικού σήματος (β) και του ανασχηματισμού του δειγματοληπτικού σήματος στο μισό του αρχικού ρυθμού δειγματοληψίας.

Σύμφωνα με το θεώρημα δειγματοληψίας Nyquist–Shannon [1], [2], το οποίο ουσιαστικά αποτελεί μία γέφυρα μεταξύ των συνεχών και των διακριτών σημάτων, ένα αναλογικό σήμα μπορεί να αναπαραχθεί από το αντίστοιχο διακριτό όταν έχει

χρησιμοποιηθεί συχνότητα δειγματοληψίας η οποία είναι τουλάχιστον διπλάσια από την μέγιστη συχνότητα του αρχικού σήματος.

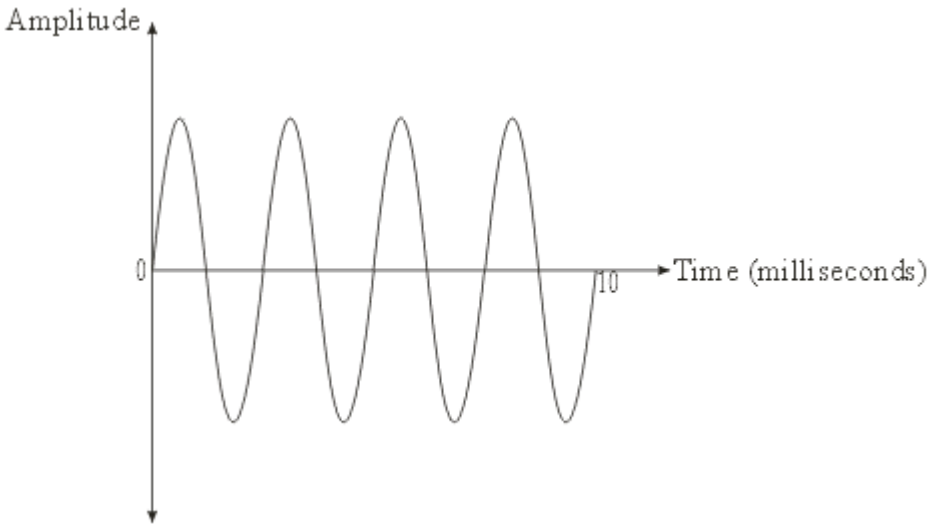
Για παράδειγμα, αν οι συχνότητες ενός συνεχούς σήματος εκτείνονται μέχρι τα 10000Hz, τότε πρέπει να χρησιμοποιηθεί συχνότητα δειγματοληψίας τουλάχιστον ίση με 20KHz. Συνηθισμένες συχνότητες δειγματοληψίας για την περίπτωση των ηχητικών σημάτων είναι 44.1KHz για μουσική και 32, 16 ή και 8KHz για ομιλία.

Επιπλέον, αν και μεγάλες συχνότητες δειγματοληψίας βελτιώνουν την ποιότητα του ψηφιακού σήματος, μπορεί να οδηγήσουν σε μεγάλες υπολογιστικές πολυπλοκότητες στους αλγορίθμους ανάλυσης, καθώς αυξάνουν το πλήθος δειγμάτων ανά δευτερόλεπτο προς επεξεργασία.

Μετά τη διακριτοποίηση του σήματος, πολλές φορές ακολουθεί η διαδικασία του φιλτραρίσματος, η οποία έχει σκοπό να αφαιρέσει ανεπιθύμητα στοιχεία ή χαρακτηριστικά από το σήμα. Συνήθως, εφαρμόζεται ένα φίλτρο προκειμένου να αφαιρεθούν ορισμένες συχνότητες ή ζώνες συχνοτήτων.

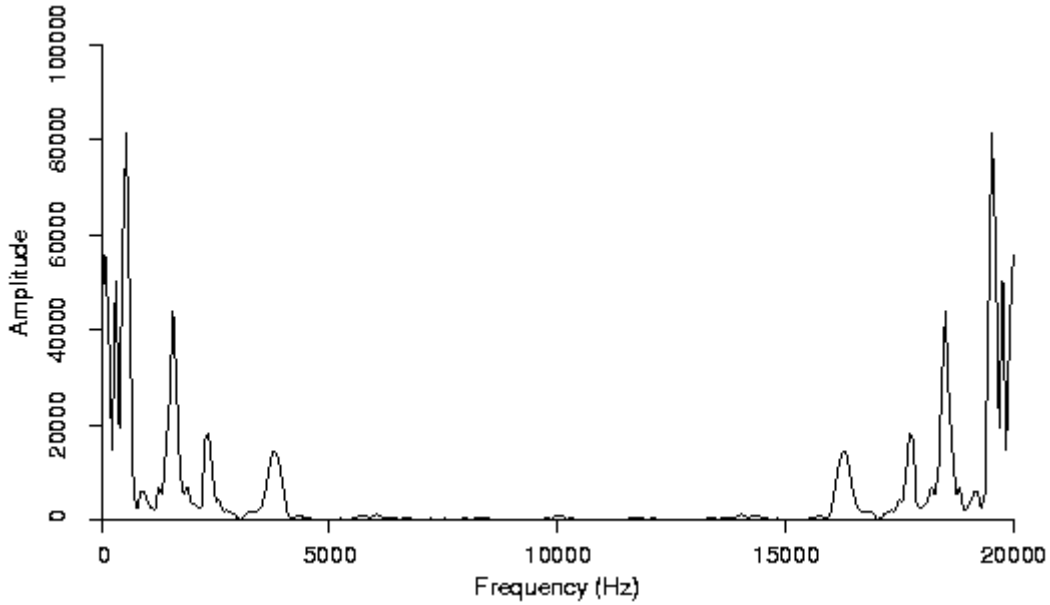
Έπειτα, ακολουθεί η εξαγωγή χαρακτηριστικών (feature extraction) από το σήμα, τα οποία αποτελούν μετέπειτα και την είσοδο στο μοντέλο προς εκπαίδευση. Υπάρχουν 3 βασικές τρόποι εξαγωγής χαρακτηριστικών ενός σήματος: αναπαράσταση του σήματος στο πεδίο του χρόνου (time domain), αναπαράσταση του σήματος στο πεδίο της συχνότητας (frequency domain) και το φασματογράφημα (spectrogram).

Στην πρώτη περίπτωση το ηχητικό σήμα αναπαριστάται από το πλάτος συναρτήσει του χρόνου, δηλαδή αποτελεί ένα διάγραμμα πλάτους-χρόνου. Ουσιαστικά, τα πλάτη που καταγράφονται σε διαφορετικά χρονικά διαστήματα αποτελούν και τα επιθυμητά χαρακτηριστικά που εξάγονται.



Σχήμα **1.3:** Αναπαράσταση ενός ηχητικού σήματος στο πεδίο του χρόνου.

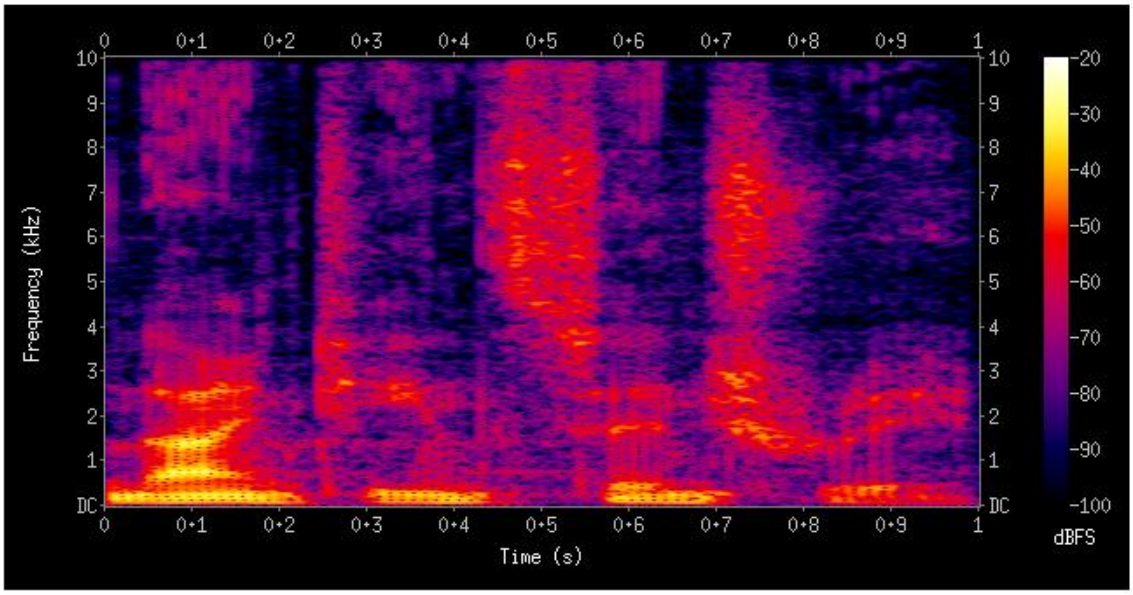
Στη δεύτερη περίπτωση το ηχητικό σήμα αναπαριστάται από το πλάτος συναρτήσει της συχνότητας, δηλαδή αποτελεί ένα διάγραμμα πλάτους-συχνότητας. Ουσιαστικά, τα πλάτη που καταγράφονται σε διαφορετικές συχνότητες αποτελούν και τα επιθυμητά χαρακτηριστικά που εξάγονται.



Σχήμα **1.4:** Αναπαράσταση ενός ηχητικού σήματος στο πεδίο της συχνότητας.

Ο περιορισμός στις δύο προαναφερθείσες περιπτώσεις είναι ότι η μία αγνοεί την πληροφορία της άλλης, δηλαδή η ανάλυση στο πεδίο του χρόνο αγνοεί εντελώς τη

συνιστώσα της συχνότητας και αντίστοιχα η ανάλυση στο πεδίο της συχνότητας αγνοεί εντελώς τη συνιστώσα του χρόνου. Υπάρχει, όμως, η δυνατότητα απεικόνισης της συχνότητας συναρτήσει του χρόνου, μέσω ενός φασματογραφήματος. Το φασματογράφημα αποτελεί ουσιαστικά ένα διάγραμμα συχνότητας-χρόνου, το οποίο αναπαριστά το πλάτος μία συγκεκριμένης συχνότητας σε ένα συγκεκριμένο χρόνο με χρώμα.



Σχήμα **1.5:** Φασματογράφημα των αγγλικών λέξεων “nineteenth century”.

1.2 Τεχνητά Νευρωνικά Δίκτυα (Neural Networks - NN)

Η ιδέα των τεχνητών νευρωνικών δικτύων (Artificial Neural Networks) άρχισε να αναπτύσσεται την δεκαετία του ‘50 από τον Frank Rosenblatt [3], με επιρροές από προηγούμενες έρευνες των Warren McCulloch και Walter Pitts [4], ο οποίος εφηύρε το νευρώνα Perceptron.

Ο Perceptron μπορεί να χαρακτηριστεί ως ένα πολύ απλό μοντέλο νευρωνικού δικτύου και αποτελεί έναν δυαδικό ταξινομητή (binary classifier):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

Όπου w είναι ένα διάνυσμα από βάρη με πραγματικές τιμές και w·x είναι το εσωτερικό γινόμενο μεταξύ των διανυσμάτων w και x. Το b είναι το 'bias', ένας σταθερός όρος ο οποίος δεν εξαρτάται από καμία τιμή εισόδου.

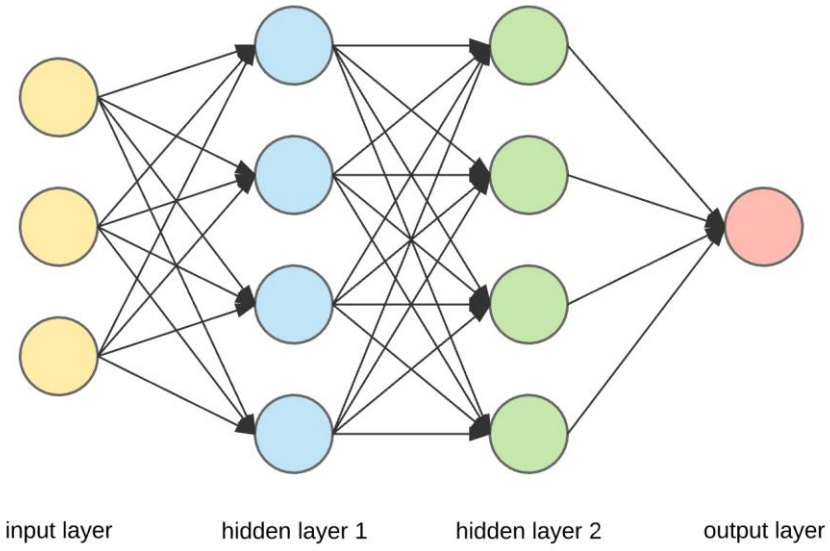
Αν και οι νευρώνες Perceptron φάνηκαν πολλά υποσχόμενοι στην αρχή, εν τέλει αποδείχθηκε ότι δεν μπορούν να εκπαιδευθούν για να αναγνωρίζουν πολλές κατηγορίες προτύπων, με αποτέλεσμα η έρευνα στα νευρωνικά δίκτυα να τελματώσει για πολλά χρόνια. Έπειτα, δημιουργήθηκαν τα νευρωνικά δίκτυα με δύο ή περισσότερα επίπεδα (αποκαλούμενα επίσης ως πολυεπίπεδα ή πολυστρωματικα νευρωνικά δίκτυα) και τα οποία διαθέτουν πολύ περισσότερη επεξεργαστική ισχύ από ένα Perceptron ενός επιπέδου.

Τα νευρωνικά δίκτυα είναι υπολογιστικά συστήματα εμπνευσμένα από τα βιολογικά νευρωνικά δίκτυα από τα οποία αποτελείται ο εγκέφαλος, και αποτελούν μια αρκετά απλουστευμένη εκδοχή αυτού. Αυτά τα συστήματα ‘‘μαθαίνουν’’ να εκτελούν ενέργειες για τις οποίες δεν έχουν άμεσα προγραμματιστεί, βασιζόμενα σε ένα πλήθος παραδειγμάτων που έχει αρχικά δοθεί.

`

1.2.1 Πλήρως Συνδεδεμένα Νευρωνικά Δίκτυα (Fully Connected Neural Networks - NN)

Τα πλήρως συνδεδεμένα νευρωνικά δίκτυα (ή dense neural networks) αποτελλούν την βασικότερη εκδοχή των νευρωνικών δικτύων. Η αρχιτεκτονική τους είναι fully connected, δηλαδή κάθε νευρώνας ενός layer συνδέεται με όλους τους νευρώνες των γειτονικων layers.



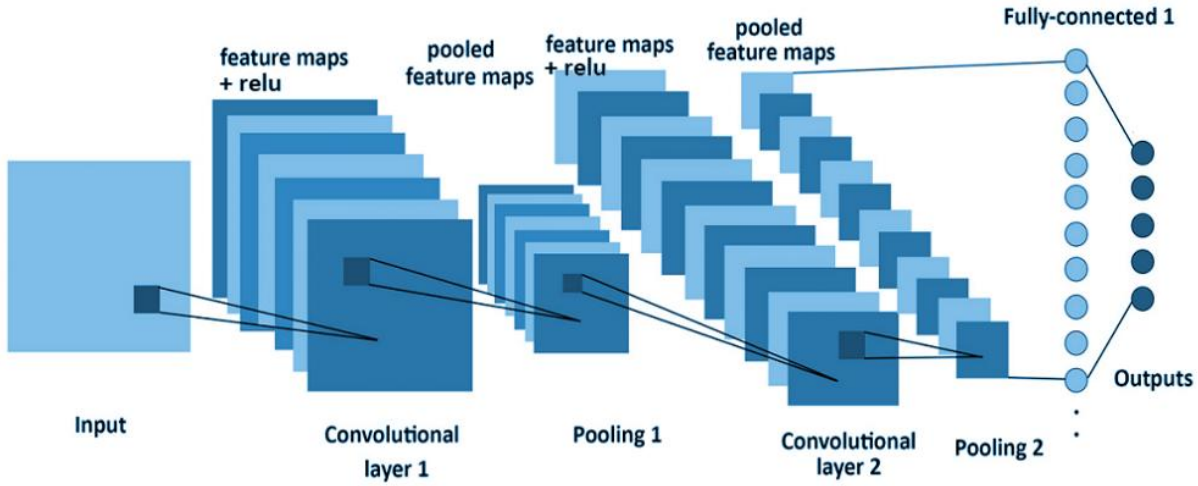
Σχήμα **1.6:** Νευρωνικό δίκτυο με δύο κρυφά επίπεδα.

Η γενική ιδέα της λειτουργίας ενός τέτοιου δικτύο αποτελείται από τις εξής βασικές έννοιες:

- Feed-forward: Το δίκτυο δέχεται σαν είσοδο το σύνολο δεδομένων (dataset) με το οποίο θα εκπαιδευτεί και υπολογίζει την έξοδο του μέσω των παραμέτρων του δικτύου. Η είσοδος του δικτύου αρχικά υπόκειται σε μια προ-επεξεργασία ώστε η μορφή των δεδομένων να ικανοποιεί τις ανάγκες του νευρωνικού. Οι παράμετροι αποτελλούνται από τις ακμές (weights) και τους νευρώνες (biases) ανα layer. Πάνω σε αυτές θα εφαρμοστεί η εκπαίδευση, και μαζί με την συνάρτηση ενεργοποίησης σε κάθε εξόδο του κάθε layer, καταλήγουμε στην έξοδο του δικτύου. Η έξοδος είναι και η εκτίμηση (predict) της αρχικής εισόδου.
- Cost function: Στην έξοδο του δικτύου, κατά την φάση της εκπαίδευσης, εφαρμόζεται μια συνάρτηση κόστους η οποία υπολογίζει το σφάλμα της εκτίμησης του δικτύου για όλα τα δεδομένα με τα οποία εκπαιδεύεται.
- Back propagation: Με την τεχνική της οπισθοδιάδοσης, και βάση του σφάλματος το οποίο βρέθηκε στο προηγούμενο στάδιο, οι παράμετροι “ρυθμίζονται” και η όλη διαδικασία επαναλαμβάνεται έως το σφάλμα να ελαχιστοποιηθεί. Κάθε τέτοια επανάληψη ορίζεται ως epoch. Στην οπισθοδιάδοση (όπως και στην συνάρτηση σφάλματος) μπορούν να υλοποιηθούν αρκετές εναλλακτικές τεχνικές, όπως Gradient Descent [5], Adam [6], RMSprop [7], κτλ.

1.2.2 Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks - CNN)

Το συνελικτικό νευρωνικό δίκτυο αποτελεί μια διαφορετική αρχιτεκτονική δικτύου που χρησιμοποιείται ιδιαίτερα στην ανάλυση και αναγνώριση εικόνας.



Σχήμα **1.7:** Δομή ενός συνελικτικού νευρωνικού δικτύου.

Μια βασική διαφορά των convolutional neural network από τα κλασσικά feed forward νευρωνικά είναι η ανοχή τους στα μη προ-επεξεργασμένα δεδομένα. Χωρίς δηλαδή το κατάλληλο preprocessing, τα convolutional καταφέρνουν και ανιχνεύουν τα χαρακτηριστικά των δεδομένων. Αυτό γίνεται τροποποιώντας τα δεδομένα σε μορφή πολύ πιο εύκολη προς επεξεργασία χωρίς να χάνουν σημαντικά features των δεδομένων, κάτι που είναι πολύ σημαντικό για την πρόβλεψη. Μετά την επεξεργασία αυτών των δεδομένων, καταλήγουμε σε ένα dense νευρωνικό που κάνει και την τελική πρόβλεψη των δεδομένων.

Η βασική αρχιτεκτονική των συνελικτικών (convolutional) νευρωνικών δικτύων αναλύεται σε:

Convolutional layer: Γίνεται συνέλιξη της εισόδου με φίλτρα τα οποία έχουν προκύψει από την διαδικασία εκπαίδευσης.

Activation layer: Γίνεται η γραμμικοποίηση της εξόδου του συνελικτικού επιπέδου, μέσω της συνάρτησης Rectified Linear Unit (ReLU).

Pooling layer: Σε αυτό το στάδιο γίνεται υπο-δειγματοληψία των δεδομένων συνοψίζοντας τις εξόδους γειτονικών γκρουπ νευρώνων εντός ενός παραθύρου με μια αντιπροσωπευτική τιμή, μειώνοντας έτσι το μέγεθος των δεδομένων. Οι επικρατέστερες

κατηγορίες του pooling είναι το max, sum και average. Στην παρούσα εργασία χρησιμοποιείται η max.

Fully connected layer: Κάθε νευρώνας του προηγούμενου layer ενώνεται με τον επόμενο και στην συνέχεια παράγονται οι έξοδοι του νευρωνικού.

Κεφάλαιο 2:

Σχεδιασμός και Μοντελοποίηση

Τα τεχνητά νευρωνικά δίκτυα διαφοροποιούνται ανάλογα με την δομή και την λογική με την οποία υλοποιούνται. Στην παρούσα εργασία αναλύονται και χρησιμοποιούνται δυο διαφορετικές αρχιτεκτονικές νευρωνικών δικτύων, το πλήρως συνδεδεμένο νευρωνικό δίκτυο (fully connected neural network) και το συνελικτικό νευρωνικό δίκτυο (convolutional neural network). Ο λόγος που αναπτύχθηκαν δύο μοντέλα αντί ενός είναι ότι σκοπός δεν ήταν μόνο η επίτευξη του καλύτερου αποτελέσματος, αλλά και η διερεύνηση της γνώσης για την αποδοτικότητα των νευρωνικών δικτύων σε εφαρμογές αναγνώρισης ομιλίας και ήχου.

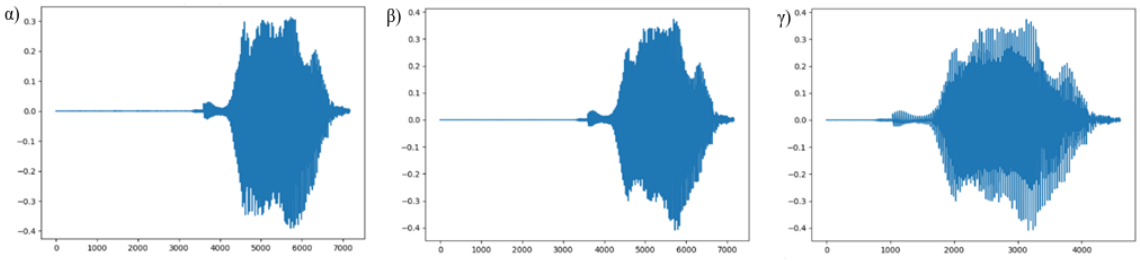
Στην εργασία υλοποιήθηκαν αρκετές παραλλαγές νευρωνικών δικτύων μέχρι να επιλεγεί η κατάλληλη δομή της κάθε αρχιτεκτονικής. Το κάθε μοντέλο δέχεται σαν είσοδο όλα τα αρχεία ήχου προς εκπαίδευση. Τα δεδομένα του κάθε αρχείου, πριν την είσοδο τους στο νευρωνικό δίκτυο, περνούν από μια διαδικασία προεπεξεργασίας (preprocessing), η οποία διαφοροποιείται ανάλογα με την αρχιτεκτονική του δικτύου.

Συγκεκριμένα, στην είσοδο του συνελικτικού δικτύου τα καλύτερα αποτελέσματα επήλθαν στην περίπτωση που εφαρμόστηκαν μόνο αλλαγή ρυθμού δειγματοληψίας, filtering και trimming, ενώ για το πλήρως συνδεδεμένο δίκτυο εφαρμόστηκαν επιπλέον διάφοροι αλγόριθμοι για την εξαγωγή χαρακτηριστικών, όπως Fourier και ο MFCC (Mel-Frequency Cepstral Coefficients) σε συνδυασμό με stretching. Η μεγαλύτερη επιτυχία εκπαίδευσης επιτυγχάθηκε μέσω του αλγορίθμου MFCC.

Οι δομές οι οποίες ερευνήθηκαν ποικίλουν στον αριθμό των νευρώνων ανά layer, στον αριθμό των layer, στην συνάρτηση κόστους αλλά και στον αλγόριθμο του back propagation.

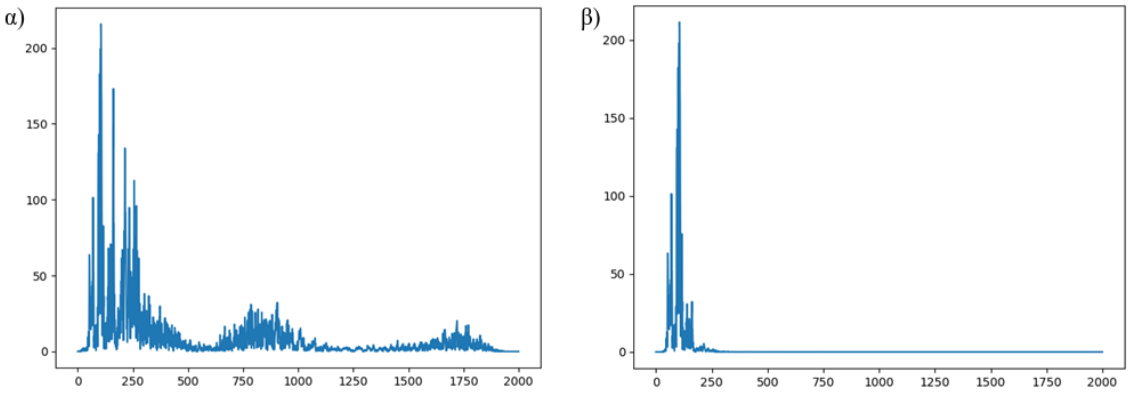
2.1 Προεπεξεργασία δεδομένων

Προτού αρχίσει το στάδιο αναγνώρισης, το σήμα προεπεξεργάζεται με κατάλληλες μεθόδους. Συγκεκριμένα, εφαρμόζονται οι διαδικασίες αλλαγής ρυθμού δειγματοληψίας στα 8000Hz, filtering, trimming και stretching σε όλα τα δεδομένα πρωτού εισαχθούν στο μοντέλο. Αυτό πραγματοποιείται ως διαδικασία και για τα δύο μοντέλα που αναπτύχθηκαν, και για το πλήρως συνδεδεμένο και για το συνελικτικό.



Σχήμα 2.1: Απεικόνιση σήματος εισόδου πριν (α) και μετά (β) την εφαρμογή φίλτρου.

Αναλυτικότερα, σε όλα τα αρχεία εισόδου εφαρμόστηκε φίλτρο ώστε να μειωθούν τα επίπεδα του θορύβου και ταυτόχρονα να διατηρηθεί η πληροφορία του σήματος που είναι επιθυμητό να αναλυθεί. Με με την εφαρμογή φίλτρου διατηρούνται τα δεδομένα του σήματος που ανήκουν στην τυπική ζώνη συχνοτήτων που σχετίζονται με τον μέσο άνθρωπο. Ένας ενήλικος άνδρας παράγει ήχους από 85 έως 180Hz, ενώ μία γυναίκα από 165 έως 255Hz. Παρότι οι άνθρωποι όταν παράγουν αυτές τις συχνότητες παράλληλα παράγουν και ανώτερες αρμονικές τους συχνότητες δεν κρίθηκε σκόπιμο αυτές να διατηρηθούν καθώς σκοπός δεν είναι η πιστότερη αναπαραγωγή ήχου, αλλά η αναγνώριση και κατηγοριοποίησή του. Το φίλτρο που εφαρμόστηκε είναι bandpass με σημεία αποκοπής τα 85-255 Hz και είναι τύπου IIR butterworth.



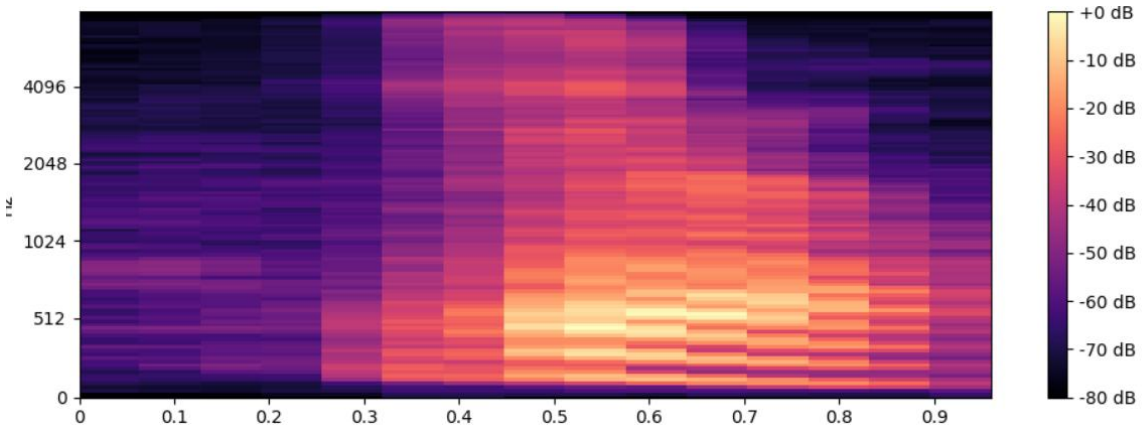
Σχήμα 2.2: Απεικόνιση σήματος εισόδου πριν (α) και μετά (β) την εφαρμογή φίλτρου.

Επιπλέον, με την εφαρμογή του trimming αφαιρούνται τυχόν παύσεις πριν ή μετά την εκφώνηση των ψηφίων και έτσι το σύστημα διατηρεί μόνο την χρήσιμη πληροφορία για την αναγνώριση ομιλίας. Συγκεκριμένα, εξασφαλίζεται ότι οι φθόγγοι που προφέρονται από τον κάθε ομιλητή θα βρεθούν σε χρονικό σημείο όσο το δυνατό πιο κοντινό μεταξύ των διαφορετικών δεδομένων εισόδου. Ουσιαστικά με αυτή την επεξεργασία εξάγονται οι σχετικές θέσεις τόσο των αυτούσιων σημείων του σήματος (raw audio ή fourier) όσο και του spectrogram και των features. Εάν δεν εφαρμόζοταν trimming οι θέσεις των patterns θα βρίσκονται σε διαφορετικά σημεία στον χρόνο (άξονας X) με αποτέλεσμα να προσθέσουν θόρυβο στην διαδικασία.

Τέλος, στα σήματα εισόδου του πλήρως συνδεδεμένου νευρωνικού δικτύου εφαρμόζεται επιπλέον και stretching, το οποίο αναπτύσσεται πιο λεπτομερώς στην επόμενη υποενότητα.

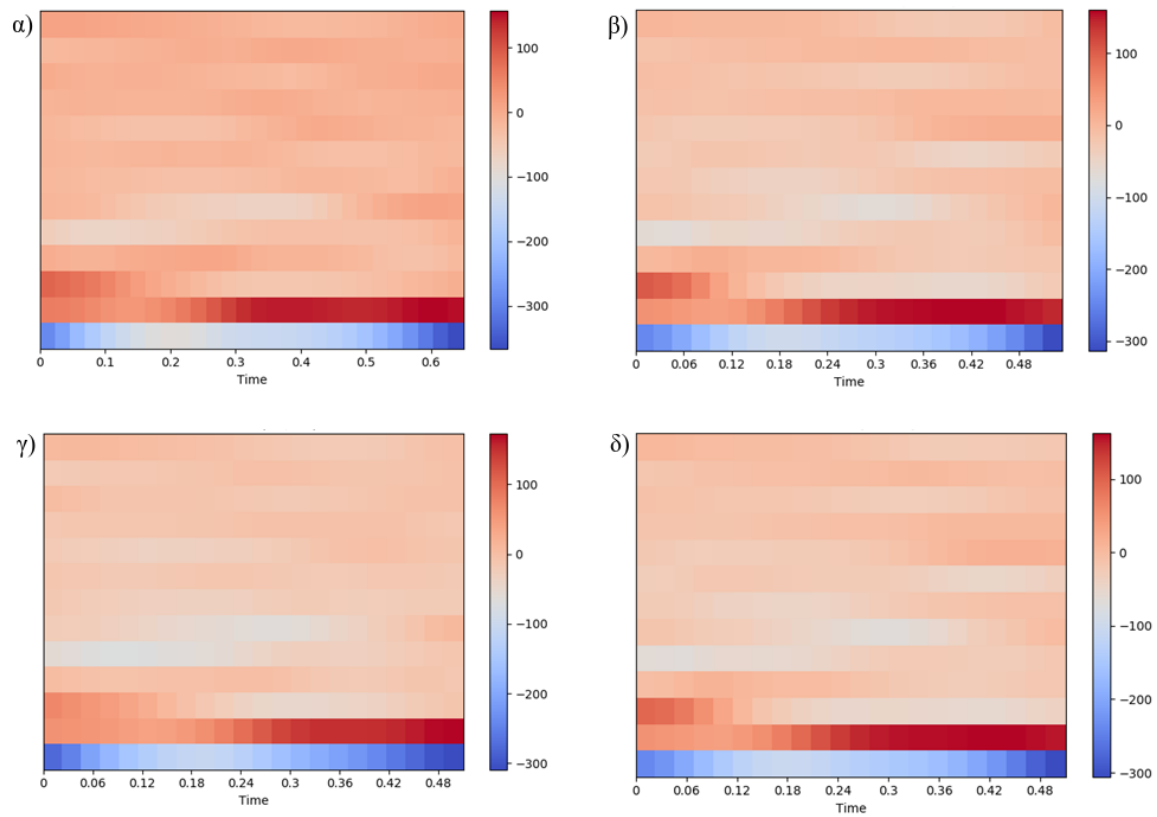
2.2 Εξαγωγή χαρακτηριστικών (feature extraction)

Το cepstrum ενός σήματος προκύπτει αν πάρουμε τον λογάριθμο του magnitude του μετασχηματισμού Φουριέ ενός σήματος. Το Mel cepstrum ουσιαστικά είναι η μεταφορά του cepstrum σε κλίμακα συχνοτήτων πιο κοντινή με το πως αντιλαμβάνεται ο άνθρωπος τις διαφορές στην συχνότητα.



Σχήμα 2.3: Φασματογράφημα Mel-Frequency.

Στις δοκιμές που πραγματοποιήθηκαν, επιχειρήθηκε να εκπαιδευτεί ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο με τα MFCCs των ηχητικών αρχείων, εξάγοντας features από το cepstrum. Η ιδέα ήταν να συσχετίζονται τα σήματα τόσο στον άξονα του χρόνου όσο και σε αυτόν της συχνότητας στην μορφή του cepstrum.



Σχήμα 2.4: Απεικόνιση σήματος εισόδου πριν (α) και μετά (β) την εφαρμογή φίλτρου.

Το stretching φάνηκε να επηρεάζει περισσότερο τη μέθοδο εκπαίδευσης με MFCCs features. Εφαρμόζοντας stretching μετά το trimming οι τιμές που επιστρέφει ο αλγόριθμος MFCC είναι πολύ πιο κοντά μεταξύ τους σε σχέση με το αρχικό σήμα. Το αποτέλεσμα της εφαρμογής του stretching είναι η χρονική διάρκεια του σήματος να είναι ίδια για όλα τα δεδομένα και με αυτό τον τρόπο επιτρέπεται στο νευρωνικό δίκτυο να εντοπίσει με μεγαλύτερη ευκολία patterns. Στην περίπτωση που δεν εφαρμοστεί stretching, τα patterns στον άξονα X έχουν μεγαλύτερο μήκος ανάλογα με την ταχύτητα που εκφωνούνται από τον κάθε ομιλητή. Με αυτή την μέθοδο τα δεδομένα εισόδου κανονικοποιούνται χωρίς να χάνεται χρήσιμη πληροφορία για την ταξινόμηση τους.

Παρατηρώντας το σχήμα των patterns χωρίς stretching προκύπτει ότι το στοιχείο στο χρονικό σημείο 0.2 διαθέτει σημαντική διαφορά στον χρωματισμό του, ενώ στην δεύτερη περίπτωση ο χρωματισμός είναι ο ίδιος. Τέλος παρατηρείται ότι ο αλγόριθμός του stretch κανονικοποιεί τα δεδομένα και ως προς το magnitude.

Για να επηρεαστούν όσο το δυνατόν λιγότερο τα στοιχεία των αρχείων υπολογίζεται η μέση διάρκεια ενός clip ήχου και προσαρμόζονται όλα τα clips στην διάρκεια αυτή.

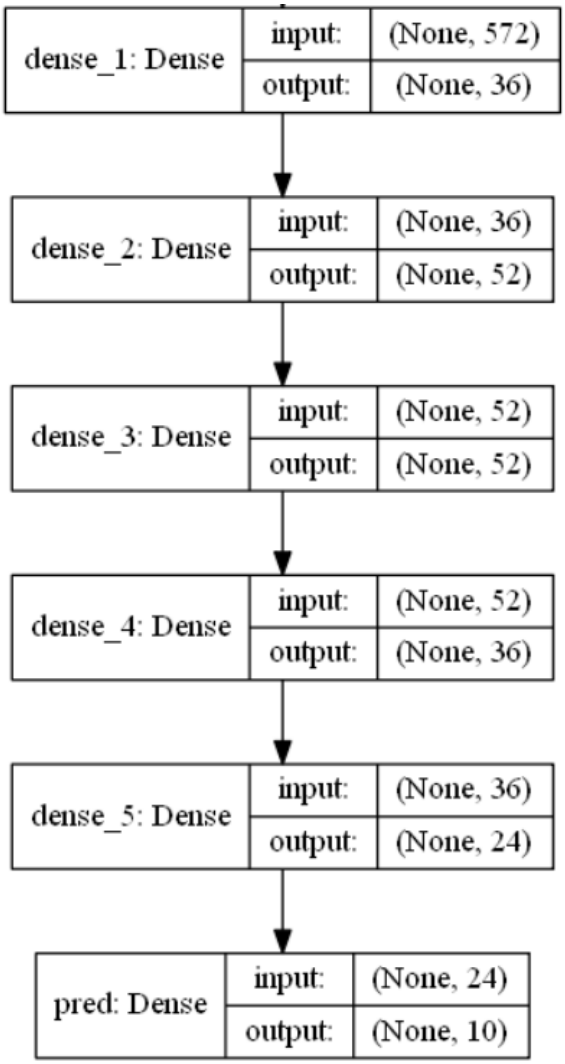
Επιπλέον, δοκιμάστηκε να εξαχθούν οι κορυφές του διαγράμματος Φουριέ των σημάτων και να δημιουργηθεί ένα feature set με ζεύγη διαφοράς μεταξύ των συχνοτήτων των κορυφών. Ο στόχος ήταν να εξουδετερωθεί η διαφορά στην τονικότητα μεταξύ των ομιλιτών και να δημιουργηθεί στο τέλος μια συχνοτική ταυτότητα για το κάθε ψηφίο ώστε να καταταχθεί από το νευρωνικό δίκτυο.

Για την εφαρμογή της μεθόδου υπολογίστηκαν τα πρώτα 50 peaks του 1ου τεταρτημόριου, κατατάχθηκαν με βάση το magnitude και έπειτα υπολογίστηκαν οι διαφορές των συχνοτήτων ανά δύο μεταξύ τους. Τα αποτελέσματα ήταν έντονα επηρεασμένα από background θορύβους που δεν έγινε εφικτό να εξουδετερωθούν με την εφαρμογή του φίλτρου με συνέπεια η ταξινόμηση να μην είναι αποδοτική.

2.3 Δομή

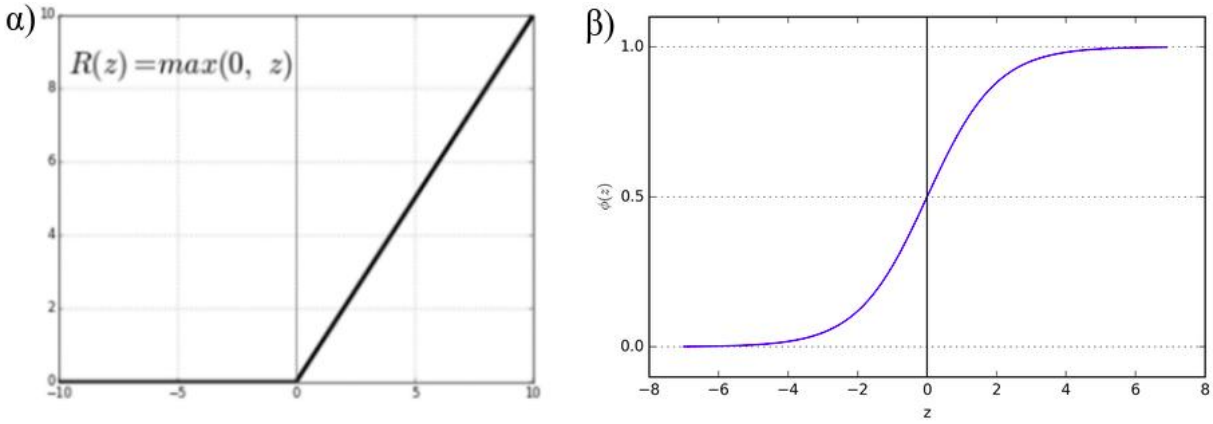
2.3.1 Πλήρως συνδεδεμένο Νευρωνικό Δίκτυο

Η τελική μορφή του πλήρως συνδεδεμένου νευρωνικού δικτύου αποτελλείται από 572 νευρώνες στο επίπεδο εισόδου, 5 κρυφά επίπεδα με 36, 52, 52, 36 και 24 νευρώνες αντίστοιχα και τέλος, το επίπεδο εξόδου διαθέτει 10 νευρώνες (όσες και οι πιθανές κατηγορίες ταξινόμησης).



Σχήμα 2.5: Δομή του πλήρως συνδεδεμένου μοντέλου μου αναπτύχθηκε.

Ως συνάρτηση ενεργοποίησης σε κάθε κρυφό layer χρησιμοποιήθηκε η ReLU, ενώ στο επίπεδο εξόδου έγινε χρήση της συνάρτησης softmax, καθώς η τελική έξοδος αποτελείται από 10 εξόδους.



Σχήμα 2.6: Γραφική απεικόνιση της συνάρτησης ενεργοποίησης ReLU (α) και της συνάρτησης softmax (β).

Η ReLU είναι μια υπολογιστικά εύκολη συνάρτηση ενεργοποίησης όπου είναι γραμμική για όλες τις θετικές τιμές εξόδου και μηδέν για όλες τις αρνητικές. Στο τελικό layer, εφαρμόστηκε η συνάρτηση ενεργοποίησης softmax, όπου δέχεται ένα vector K αριθμών (10 στο παρόν πρόβλημα προς επίλυση) και το κάνει normalize σε μια κατανομή πιθανοτήτων.

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

Ως συνάρτηση κόστους χρησιμοποιήθηκε η cross-entropy.

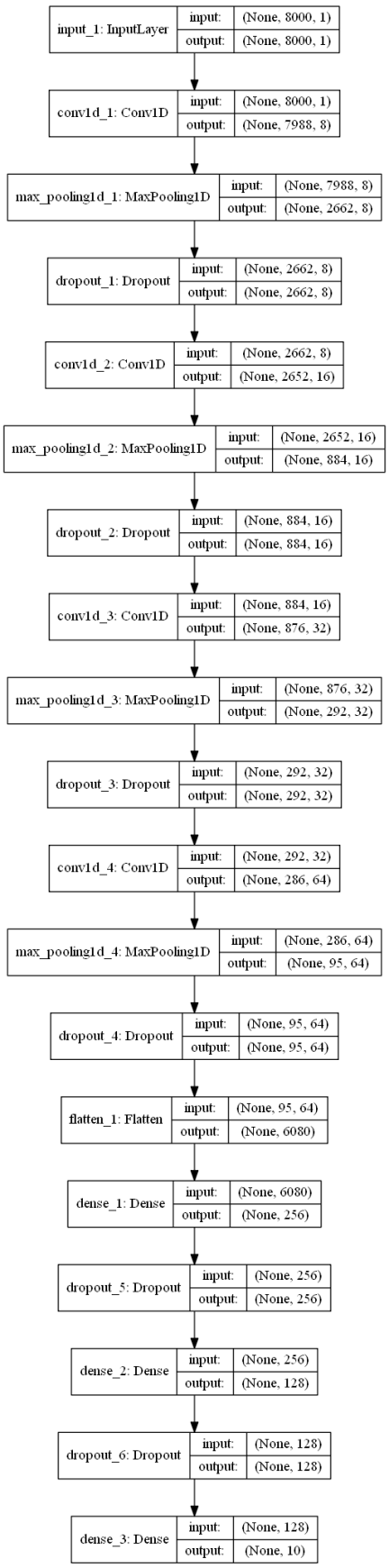
$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)]$$

Τέλος για την οπισθοδιάδοση έγινε χρήση του αλγορίθμου RMSprop. Αντίστοιχα αποτελέσματα υπήρξαν και με την χρήση του αλγορίθμου Adam, που αποτελεί μια βελτιωμένη έκδοση του gradient descent.

Η διαδικασία μάθησης έχει οριστεί στα 100 epochs με δυνατότητα διακοπής εάν το accuracy δεν βελτιώνεται για πάνω από 10 epochs. Με αυτή την τεχνική διατηρούνται οι παράμετροι της καλύτερης εποχής και μειώνεται αισθητά ο χρόνος εκπαίδευσης του δικτύου.

2.3.2 Συνελικτικό Νευρωνικό Δίκτυο

Η δομή του συνελικτικού μοντέλου είναι αρκετά πιο πολύπλοκή και παρουσιάζεται στο Σχήμα 2.7.

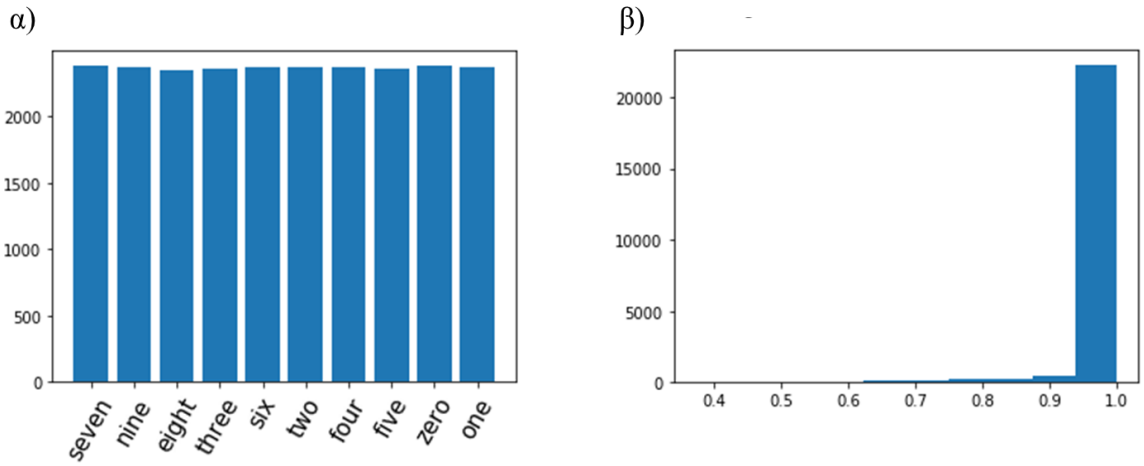


Σχήμα 2.7: Δομή του συνελικτικού μοντέλου μου αναπτύχθηκε.

Κεφάλαιο 3:

Αποτελέσματα

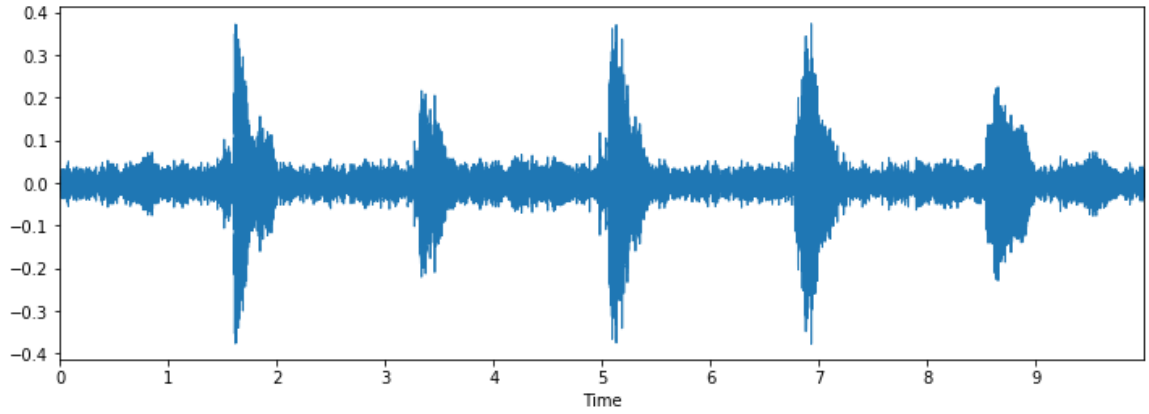
Για την εκπαίδευση του μοντέλου χρησιμοποιήθηκε ένα σύνολο δεδομένων με εγγραφές εκφωνούμενων ψηφίων από 997 διαφορετικούς ομιλητές [11]. Συγκεκριμένα, το σύνολο περιλαμβάνει 23666 εγγραφές, από τις οποίες 80% χρησιμοποιήθηκε για εκπαίδευση και 20% για αξιολόγηση του μοντέλου. Το σύνολο διαθέτει 10 κατηγορίες (classes) που αντιστοιχούν στα ψηφία από το 0 μέχρι και το 9.



Σχήμα 3.1: Γραφική αναπαράσταση του πλήθους κάθε κατηγορίας (α) και της διάρκειας (β) του συνόλου δεδομένων που χρησιμοποιήθηκε για εκπαίδευση και αξιολόγηση.

Επιπλέον, εκτός από την αξιολόγηση στα δεδομένα του test set, χρησιμοποιήθηκαν και μία ηχογράφηση γυναικείας φωνής, προκειμένου να δοκιμαστεί περεταίρω τα μοντέλα.

Η απεικόνιση αυτών του ηχητικού παρουσιάζεται στο Σχήμα 3.2, όπου μπορούν εύκολα να διακριθούν οι παύσεις και αντίστοιχα τα σημεία εκφώνησης ψηφίων.



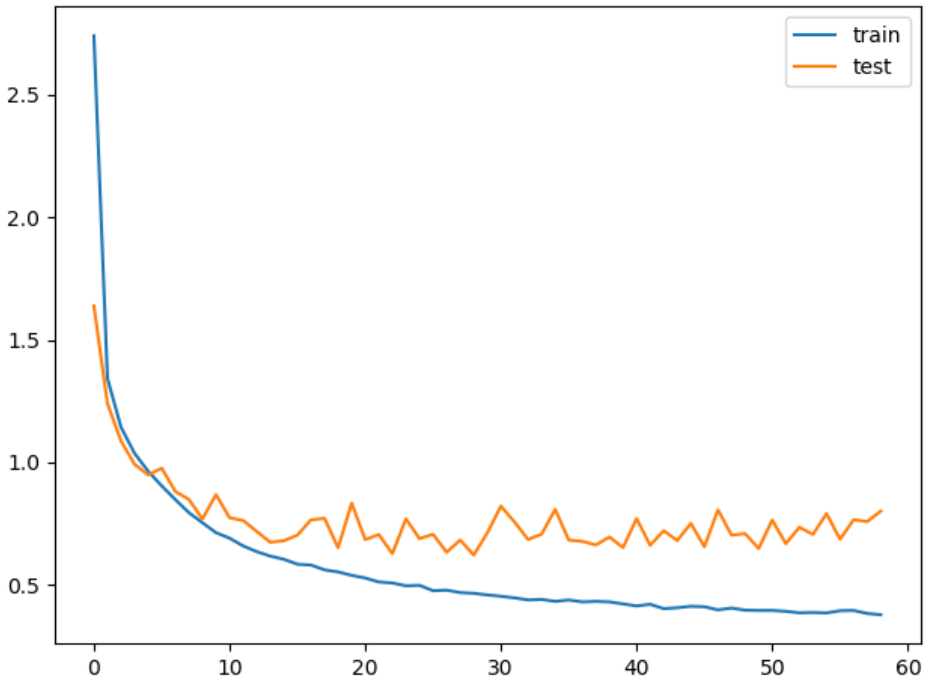
Σχήμα 3.2: Γραφική απεικόνιση της εκφώνησης των ψηφίων 7, 2, 4, 1 και 9.

3.1 Πλήρως συνδεδεμένο Νευρωνικό Δίκτυο

Τα αποτελέσματα που εξάχθηκαν με είσοδο τη διακριτοποίηση της συχνότητας των αρχείων ήχου στα 8000 δείγματα δεν ήταν τόσο καλά. Κάνοντας δοκιμές στην επεξεργασία των δεδομένων μέσω των αλγορίθμων Fourier και MFCC, τα αποτελέσματα βελτιώθηκαν. Συγκεκριμένα, κάνοντας preprocessing με τον αλγόριθμο MFCC, επιτεύχθηκε accuracy του train στο 87% και του test στο 81% με προοπτικές βελτίωσης.

```
Accuracy: 87.90
Audio: seven  and predicted: seven
Audio: five  and predicted: five
Audio: nine  and predicted: nine
Audio: six  and predicted: six
Audio: nine  and predicted: three
Audio: three  and predicted: three
Audio: five  and predicted: seven
Audio: five  and predicted: five
Audio: six  and predicted: six
Audio: four  and predicted: five
Train Score:  [0.3927600352432419, 0.8789879568983732]
Validation Score:  [0.8010218094107366, 0.8111533586063312]
```

Σχήμα 3.3: Αποτελέσματα αξιολόγησης μοντέλου στο test dataset.



Σχήμα 3.4: Γραφική απεικόνιση σφάλματος κατά την διάρκεια της εκπαίδευσης σε train και test data.

Το πλήρως συνδεδεμένο μοντέλο δεν απέδωσε καθόλου καλά όταν του δόθηκε ως είσοδος η ηχογράφηση, καθώς δεν κατάφερε να βρει κανένα ψηφίο.

```
Predictions

Word #1 : three
Word #2 : five
Word #3 : three
Word #4 : six
Word #5 : five
```

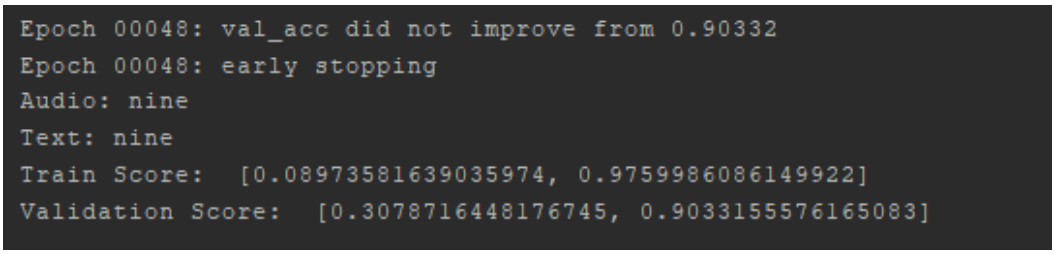
Σχήμα 3.5: Προβλέψεις πλήρους συνδεδεμένου νευρωνικού δικτύου για ηχογράφηση 5 ψηφίων.

3.2 Συνελικτικό Νευρωνικό Δίκτυο

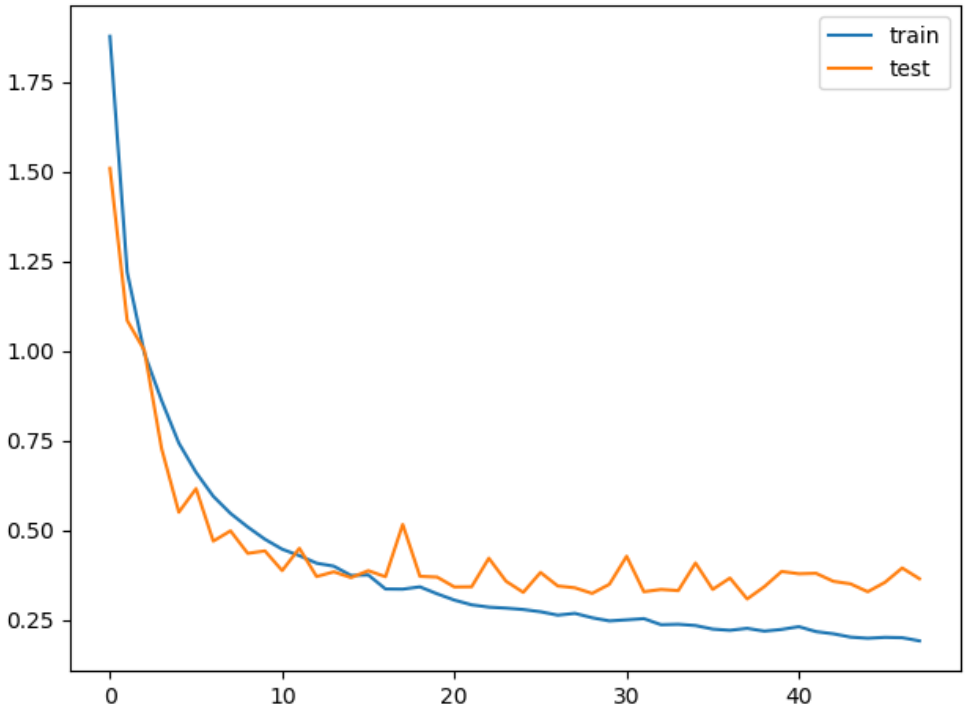
Η επιλογή αυτής της αρχιτεκτονικής βοήθησε πολύ και στην αντιμετώπιση του overfitting. Κατά την υλοποίηση του fully-connected νευρωνικού, το πρόβλημα του

overfitting παρουσιάστηκε αρκετές φορές λόγω της μη επιθυμητής αναλογίας των features των δεδομένων (8000) με τον αριθμό των δεδομένων για εκπαίδευση (23000). Κάνοντας και την χρήση του preprocessing που προαναφέρθηκε, το overfitting υπήρχε ακόμα και με τεχνικές αντιμετώπισης του όπως regularization και dropouts σε κάθε layer. Τα convolutional και pooling layers ανιχνεύουν τα high-level features της εισόδου ώστε αυτό το πρόβλημα να αποφεύγεται με επιτυχία.

Τα αποτελέσματα του σε σχέση με το dense είναι αρκετά ικανοποιητικά, καθώς φτάνουμε accuracy στο train της τάξης του 97% και 90% στο test. Η διαδικασία μάθησης είχε οριστεί στα 100 epochs, παρ'όλα αυτά έφτασε το μεγαλύτερο accuracy στα 48 epochs.



Σχήμα 3.6: Γραφική απεικόνιση σφάλματος κατά την διάρκεια της εκπαίδευσης σε train και test data.



Σχήμα 3.7: Γραφική απεικόνιση σφάλματος κατά την διάρκεια της εκπαίδευσης σε train και test data.

Σε αντίθεση με το πλήρως συνδεδεμένο νευρωνικό μοντέλο, το συνελικτικό απέδωσε εξαιρετικά στην πρόβλεψη των ψηφίων της ηχογράφησης, καθώς προέβλεψε σωστά όλα ψηφία.

```
Predictions

Word #1 : seven
Word #2 : two
Word #3 : four
Word #4 : one
Word #5 : nine
```

Σχήμα 3.8: Προβλέψεις συνελικτικού νευρωνικού δικτύου για ηχογράφηση 5 ψηφίων.

Κεφάλαιο 4:

Συμπεράσματα

Μετά από έρευνα πάνω στην ανάλυση ήχου έγινε αντιληπτό ότι χωρίς το κατάλληλο preprocessing, η επιλογή ενός dense νευρωνικού δικτύου για την αναγνώριση ήχου δεν είναι αρκετά αποτελεσματική. Το feature set που παράγεται και θέτουμε σαν είσοδο στο νευρωνικό δίκτυο είναι πολύ μεγάλο σε σχέση με το dataset μας.

Με το κατάλληλο preprocessing μπορούμε να επιτύχουμε ένα ικανοποιητικό accuracy, παρ'όλα αυτά η εύρεση του κατάλληλου αλγορίθμου εξαρτάται εκτός από τον τύπο των δεδομένων (ποιότητα ήχου, θόρυβος κτλ) και από την υπολογιστική μονάδα που θα κάνει αυτή την επεξεργασία, καθώς η μεγάλη επεξεργαστική ισχύ για την ανάλυση όλων αυτών των δεδομένων είναι αναγκαία.

Ακόμα, ενώ στο train είχαμε αρκετά καλά αποτελέσματα, στο test είχαμε τεράστια απόκλιση στην πρόβλεψη των δικών μας ηχογραφήσεων, για το λόγο ότι εξαιτίας των μεγάλων feature set παρουσιαζόταν το πρόβλημα του overfitting.

Κάνοντας χρήση του convolutional νευρωνικού δικτύου, το preprocessing γίνεται έμμεσα εντός του δικτύου. Το δίκτυο εκπαιδεύεται στο να ανιχνεύει τα "δυνατά" features των δεδομένων με την χρήση των convolutional και pooling layers, και στην συνέχεια εκπαιδεύεται με τον κλασσικό τρόπο μέσω ενός dense δικτύου.

Βιβλιογραφία

[1] Marks, R.J.(II): *Introduction to Shannon Sampling and Interpolation Theory*, Springer-Verlag, 1991.

[2] Marks, R.J.(II), Editor: Advanced Topics in Shannon Sampling and Interpolation Theory, Springer-Verlag, 1993.

[3] Rosenblatt, Frank (1957). "The Perceptron—a perceiving and recognizing automaton" Report 85-460-1. Cornell Aeronautical Laboratory.

[4] *McCulloch, Warren; Walter Pitts (1943).* "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics.. ***5*** *(4): 115–133. doi:10.1007/BF02478259.*

[5] Cauchy, Augustin. "Méthode générale pour la résolution des systemes d'équations simultanées." Comp. Rend. Sci. Paris 25.1847 (1847): 536-538.

[6] Adam optimizer: (Kingma & Ba, 2015).

[7] Geoffrey Hinton Neural Networks for machine learning online course.

[8] P. Mermelstein (1976), "Distance measures for speech recognition, psychological and instrumental," in *Pattern Recognition and Artificial Intelligence*, C. H. Chen, Ed., pp. 374–388. Academic, New York.

[9] S.B. Davis, and P. Mermelstein (1980), "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4), pp. 357–366.

[10] J. S. Bridle and M. D. Brown (1974), "An Experimental Automatic Word-Recognition System", JSRU Report No. 1003, Joint Speech Research Unit, Ruislip, England.

[11] Wolfram Research, "Spoken Digit Commands" from the Wolfram Data Repository (2018)

Παράρτημα Α

Κώδικας

- *ASR.py*

```
import os


from Deliverable import signal_processing
from Deliverable import feature_exrtaction
from Deliverable import train_model
from Deliverable import split_to_words
from Deliverable import predict

from pathlib import Path
import sounddevice as sd
from scipy.io.wavfile import write
import sys
import librosa   # for audio processing

current_directory = os.getcwd()

# Choose audio dataset
# train_audio_path = Path('../free-spoken-digit-dataset-medium')  # Audio Path
train_audio_path = Path('../speech_commands_dataset_small')   # Audio Path
# train_audio_path = Path('../speech_commands_dataset_full')   # Audio Path
train_audio_path = Path('../free-spoken-digit-dataset-medium') #Audio Path
#train_audio_path = Path('../speech_commands_dataset_full') #Audio Path

print("Training Dataset Folder:", train_audio_path)
print("Training Dataset Folder:",train_audio_path)

labels = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]
```

```
    # Training
    # rawAudioFiles,all_label, sr, meanDuration =
signal_processing.resample_dataset(labels, train_audio_path)
    #rawAudioFiles,all_label, sr, meanDuration =
signal_processing.resample_dataset(labels, train_audio_path)
    processedAudioFiles, all_labels, sr, meanDuration =
signal_processing.process_dataset(labels, train_audio_path)

    print('PreProcessing\n1)Raw Data\n2)Raw Data Stretched\n3)Fourier\n4)Fourier
Stretched\n5)MFCC\n6)Fourier Peaks\n')
    ans_pp = int(input('Select one of the above for data pre-processing\n'))
    print('Model Architecture\n1)Dense\n2)Convolutional\n')
    ans_m = int(input('Select one of the above for train\n'))

    if ans_pp == 1:
        all_features = feature_exrtaction.rawData(processedAudioFiles, sr)
    elif ans_pp == 2:
        all_features = feature_exrtaction.rawDataStretched(processedAudioFiles, sr,
meanDuration)
    elif ans_pp == 3:
        all_features = feature_exrtaction.fourier_transform(processedAudioFiles, sr)
    elif ans_pp == 4:
        all_features = feature_exrtaction.fourier_transform_stretched(processedAudioFiles,
sr, meanDuration)
    elif ans_pp == 5:
        all_features = feature_exrtaction.extract_mfccs(processedAudioFiles, sr,
meanDuration)
    elif ans_pp == 6:
        all_features = feature_exrtaction.extract_fourier_peaks(processedAudioFiles, sr,
meanDuration)
    else:
        sys.exit('Invalid selection')
```

```
    if ans_m == 1:
        model_path = train_model.train_dense(all_features, all_labels, labels,
model_name="Just_testing_dense")
    elif ans_m == 2:
        model_path = train_model.train_convolutional(all_features, all_labels, labels,
"Just_testing_conv")
    else:
        sys.exit('Invalid selection')

    test_audio_path="../free-spoken-digit-dataset-medium"

    # Predicting Features extraction
    # raw_predict_files,all_label, sr, meanDuration =
signal_processing.resample_dataset(["predict_set"], Path(test_audio_path))
    processed_predict_files, all_labels, sr, meanDuration =
signal_processing.process_dataset(labels, Path(test_audio_path))

    if ans_pp == 1:
        predict_features =feature_exrtaction.rawData(processed_predict_files, sr)
    elif ans_pp == 2:
        predict_features = feature_exrtaction.rawDataStretched(processed_predict_files, sr,
meanDuration)
    elif ans_pp == 3:
        predict_features = feature_exrtaction.fourier_transform(processed_predict_files, sr)
    elif ans_pp == 4:
        predict_features =
feature_exrtaction.fourier_transform_stretched(processed_predict_files, sr,
meanDuration)
    elif ans_pp == 5:
        predict_features = feature_exrtaction.extract_mfccs(processed_predict_files, sr,
meanDuration)
    elif ans_pp == 6:
```

```
        predict_features = feature_exrtaction.extract_fourier_peaks(processed_predict_files,
sr, meanDuration)
    else:
        sys.exit('Error')
    #raw_predict_files,all_label, sr, meanDuration =
signal_processing.resample_dataset(["predict_set"], Path(test_audio_path))
    processed_predict_files, sr, meanDuration =
signal_processing.process_predict_dataset(labels, Path(test_audio_path))



    print("Predictions")
    # Predict based on trained model
    if (ans_m==2):
        predict.predict(model_path, predict_features, processed_predict_files,"conv")
    else :
        predict.predict(model_path, predict_features, processed_predict_files, "dense")
```

● *feature_exrtaction.py*

```
import librosa
from scipy.fftpack import fft
import numpy as np


def fourierTransform(fs,y,label):
    # Number of samplepoints
    N = len(y)
    # sampling period
    T = 1.0 / fs

    yf = fft(y)

    #Extract 1st quarter of frequency domain data
    xf = np.linspace(0.0, 1.0 / (2.0 * T), N / 2) * 0.5  # No need for both semi-axis the rfequency domain in Hertz
    yf = 2.0 * np.abs(yf[:N // 2])  # The frequency magnitude

    # plt.plot(xf,yf,label=str(label) + ' - file :' + file.name +"N" + str(N)) # plot the fourier transform
    # plt.show()

    return xf, yf, N


def fourierPeaks(fs,y,label,no_of_peaks):
    # Number of samplepoints
    N = len(y)
    # sampling period
    T = 1.0 / fs

    yf = fft(y)
```

```
    xf = np.linspace(0.0, 1.0/(2.0*T), N/2) * 0.5 # No need for both semi-axis the rfequency domain in Hertz
    yf = 2.0 * np.abs(yf[:N//2]) # The frequency magnitude

    #plt.plot(xf,yf,label=str(label) + ' - file :' + file.name +"N" + str(N)) # plot the fourier transform
    #plt.show()

    # Find dominant freq peak centers and train matching them with their frequency domains
    peaks=[]

    i=0
    step=10 # Peaks greter than step Hz in distance
    for point in yf:
        peaks.append([xf[i],point])
        i+=1

    # Choose greatest magnitude
    peaks=sorted(peaks, key=lambda x: x[1],reverse=True)
    peaks=peaks[:no_of_peaks]

    peaksDiffs=[]
    i=0
    for peak in peaks:
        if i > 1:
            peaksDiffs.append(peak[0]-prev[0])
        prev=peak
        i+=1

    return xf,yf,N,peaksDiffs
```

```
    def rawData(audioStreams,sr):
        i = 0

        all_features = []
        print("Extracting Features from " + str(len(audioStreams))  + " files")
        for audio in audioStreams:
            samples = librosa.resample(audio, sr, 8000)

            if len(samples) < 8000:
                zero_padded = np.lib.pad(audio, (0, (8000 - len(samples))), 'constant',
constant_values=0)
                all_features.append(zero_padded)
            else:
                all_features.append(samples[:8000])

            i += 1
        print("The dataset has been created")
        return all_features


    def rawDataStretched(audioStreams,sr,meanDuration):
        sumdur = 0
        i = 0

        all_features = []
        print("Extracting Features from " + str(len(audioStreams))  + " files, mean
duration=" + str(meanDuration))
        for audio in audioStreams:
            samples = librosa.resample(audio, sr, 8000)
            duration = librosa.get_duration(samples)
            sumdur += duration
```

```
            ratio = duration / meanDuration
            if ratio < 0.05:
                ratio = 0.05

            # Stretrch audio to normalise spoken word duration
            audio = librosa.effects.time_stretch(samples, ratio)

            if len(audio) < 8000:
                zero_padded = np.lib.pad(audio, (0, (8000 - len(audio))), 'constant',
constant_values=0)
                all_features.append(zero_padded)
            else:
                all_features.append(audio[:8000])



            i += 1
        print("The dataset has been stretched to the mean duration of the tracks, without
affecting pitch")
        return all_features


    def extract_mfccs(audioStreams,sr,meanDuration):
        sumdur=0
        i=0
        fsizeEQ = 0
        fsizeL = 0
        fsizeG = 0
        all_features = []
        desiredNoOfFeatures=13
        # Features Extraction an time normalisation
        print("Extracting Features from " + str(len(audioStreams)) + " files, mean
duration="+str(meanDuration))
        for audio in audioStreams:
```

```
            #sd.play(audio, sr)
            #status = sd.wait()

            duration = librosa.get_duration(audio)
            sumdur+=duration

            ratio = duration / meanDuration
            if ratio < 0.05:
                ratio = 0.05

            # Stretrch audio to normalise spoken word duration
            audio = librosa.effects.time_stretch(audio, ratio)

            #Extract features
            mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=desiredNoOfFeatures,
hop_length=178)  # Get specific number of components
            mfccs_flat = mfccs.flatten()

            #Normalize feature data set size
            featuresSize = desiredNoOfFeatures * 44  # mfccs.shape[1]

            if len(mfccs_flat) < featuresSize:
                zero_padded = np.lib.pad(mfccs_flat, (0, (featuresSize - len(mfccs_flat))),
'constant', constant_values=0)
                all_features.append(zero_padded)
                fsizeL += 1
            elif len(mfccs_flat) > featuresSize:
                all_features.append(mfccs_flat[:featuresSize])
                fsizeG += 1
            else:
                fsizeEQ += 1
                all_features.append(mfccs_flat)
```

```
            i+=1
        print("MFCC features have been extracted")
        return all_features


    def fourier_transform(audioStreams,sr):
        no_of_features = 3500
        sumdur = 0
        i = 0
        features_size = 0
        all_features = []
        print("Extracting Features from " + str(len(audioStreams)) )
        for audio in audioStreams:



            frequency, fourierMagnitude, sampleCount = fourierTransform(sr, audio, str(i))
            length = 3000

            if len(fourierMagnitude) < length:
                zero_padded = np.lib.pad(fourierMagnitude, (0, length -
len(fourierMagnitude)), 'constant',
                                         constant_values=0)
                all_features.append(zero_padded)

            else:
                all_features.append(fourierMagnitude[:length])

            i += 1
        print("The dataset has been converted from time space to frequency space")
        return all_features
```

```
    def fourier_transform_stretched(audioStreams,sr,meanDuration):
        no_of_features=3500
        sumdur=0
        i=0
        features_size=0
        all_features = []
        print("Extracting Features from "+str(len(audioStreams)) +" files, mean
duration="+str(meanDuration))
        for audio in audioStreams:

            duration = librosa.get_duration(audio)
            sumdur+=duration

            ratio = duration / meanDuration
            if ratio < 0.05:
                ratio = 0.05

            # Stretrch audio to normalise spoken word duration
            audio = librosa.effects.time_stretch(audio, ratio)

            frequency, fourierMagnitude, sampleCount = fourierTransform(sr, audio, str(i))
            length=3000

            if len(fourierMagnitude) < length:
                zero_padded = np.lib.pad(fourierMagnitude, (0, length -
len(fourierMagnitude)), 'constant', constant_values=0)
                all_features.append(zero_padded)

            else:
                all_features.append(fourierMagnitude[:length])



            i+=1
```

```
        print("The dataset has been converted from time space to frequency space")
        return all_features


    def extract_fourier_peaks(audioStreams,sr,meanDuration):
        sumdur=0
        i=0

        all_features = []
        desiredNoOfFeatures=13
        # Features Extraction an time normalisation
        print("Extracting Features from "+str(len(audioStreams)) +" files, mean
duration="+str(meanDuration))
        for audio in audioStreams:

            #sd.play(audio, sr)
            #status = sd.wait()

            duration = librosa.get_duration(audio)
            sumdur+=duration

            ratio = duration / meanDuration
            if ratio < 0.05:
                ratio = 0.05

            # Stretrch audio to normalise spoken word duration
            audio = librosa.effects.time_stretch(audio, ratio)

            #sd.play(audio, sr)
            #status = sd.wait()

            peaksNo=50
```

```
        frequency, fourierMagnitude, sampleCount, peaks = fourierPeaks(sr, audio, str(i),
peaksNo)

        peaks_flat=np.array(peaks).flatten()

        peaksNo=peaksNo*2

        if len(peaks_flat) < peaksNo:
            zero_padded = np.lib.pad(peaks_flat, (0, peaksNo - len(peaks_flat)), 'constant',
                                     constant_values=0)
            all_features.append(zero_padded)
        else :
            all_features.append(peaks_flat[:peaksNo])

        i+=1
    print("Fourier Peaks have been extracted")
    return all_features
```

● *predict.py*

```
from keras.models import load_model, Sequential
import numpy as np
import sounddevice as sd

def predict(model_filepath,audio_streams,processed_predict_files,mode):
    model = load_model(model_filepath)
    if (mode == "conv"):
        classes = np.load('classes_conv.npy')
    else:
        classes = np.load('classes_dense.npy')

    print(classes)
    i=0
    for audio in audio_streams:
        if(mode=="conv"):
            prob = model.predict(audio.reshape(1, len(audio), 1))
        else:
            prob = model.predict(np.array([audio, ]))
        index = np.argmax(prob[0])
        print("Word #"+str(i+1) +" : " +classes[index])
        #sd.play(processed_predict_files[i], 8000)
        #status = sd.wait()
        i+=1
```

● *predict_dataset.py*

```
import os
from pathlib import Path
import librosa   # for audio processing


from Deliverable import signal_processing
from Deliverable import feature_exrtaction
from Deliverable import predict


labels = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]
test_audio_path="../free-spoken-digit-dataset-medium"
test_audio_path="../speech_commands_dataset_full"
test_audio_path="./recordings"

model_path="./conv_Just_testing_conv.hdf5"

# Predicting Features extraction
#raw_predict_files,all_label, sr, meanDuration =
signal_processing.resample_dataset(["predict_set"], Path(test_audio_path))
processed_predict_files, sr, meanDuration =
signal_processing.process_predict_dataset(["george_set"], Path(test_audio_path))


predict_features=feature_exrtaction.rawData(processed_predict_files, sr)
#predict_features=feature_exrtaction.rawDataStretched(processed_predict_files, sr,
meanDuration)
#predict_features=feature_exrtaction.fourier_transform(processed_predict_files, sr)
#predict_features,feature_size=feature_exrtaction.fourier_transform_stretched(process
ed_predict_files, sr, meanDuration,feature_size)
#predict_features=feature_exrtaction.extract_mfccs(processed_predict_files, sr,
meanDuration)
```

```
    #predict_features=feature_exrtaction.extract_fourier_peaks(processed_predict_files,sr,
meanDuration)

    print("Predictions")
    # Predict based on trained model
    predict.predict(model_path, predict_features, processed_predict_files,"conv")
```

● *record_and_predict.py*

```
import os
from pathlib import Path
import sounddevice as sd
from scipy.io.wavfile import write
import librosa   # for audio processing

from Deliverable import signal_processing
from Deliverable import feature_exrtaction
from Deliverable import train_model
from Deliverable import split_to_words
from Deliverable import predict

# Record Audio
fs = 8000
max_seconds = 30
print("Recording Audio...")
myrecording = sd.rec( int(max_seconds * fs), samplerate=fs, channels=1)
input("Press Enter to stop recording...")
sd.stop()

write('./recordings/recording.wav', fs, myrecording)  # Save as WAV file

test_audio_path="./recordings"
# Split Audio to test
split_to_words.splitter("./recordings/recording.wav",test_audio_path+"/predict_set")

model_path="model_conv.hdf5"

processed_predict_files, sr, meanDuration =
signal_processing.process_predict_dataset(["predict_set"], Path(test_audio_path))
```

```
    #predict_features=feature_exrtaction.rawData(processed_predict_files, 8000)
    #predict_features=feature_exrtaction.rawDataStretched(processed_predict_files, sr,
meanDuration)
    predict_features=feature_exrtaction.fourier_transform(processed_predict_files, sr)
    #predict_features,feature_size=feature_exrtaction.fourier_transform_stretched(process
ed_predict_files, sr, meanDuration,feature_size)
    #predict_features=feature_exrtaction.extract_mfccs(processed_predict_files, sr,
meanDuration)
    #predict_features=feature_exrtaction.extract_fourier_peaks(processed_predict_files,sr,
meanDuration)

    print("Predictions")
    # Predict based on trained model
    predict.predict(model_path, predict_features, processed_predict_files)
```

● *signal_processing.py*

```
import os
from pathlib import Path



import matplotlib.pyplot as plt
from scipy.fftpack import fft
import sounddevice as sd

import librosa   # for audio processing
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from keras.utils import np_utils, to_categorical
from keras.layers import Dense, Dropout, Flatten, Conv1D, Input, MaxPooling1D,
SimpleRNN
from keras.callbacks import EarlyStopping, ModelCheckpoint, History
from keras.models import load_model, Sequential
import random



from scipy import signal

# The voiced speech of a typical adult male will have a fundamental frequency from
85 to 180 Hz, and that of a typical adult female from 165 to 255 Hz.
#def bandpassIIRFilter(data, fs, lowcut=300, highcut=3400, order=4):#telephony - i
xroia poy prokyptei apo tis anwteres armonikes einai mallon axristi
#def bandpassIIRFilter(data, fs, lowcut=60, highcut=800, order=4):
```

```
    def bandpassIIRFilter(data, fs, lowcut=60, highcut=2400, order=4):
        nyq = 0.5 * fs
        low = lowcut / nyq
        high = highcut / nyq
        b, a = signal.butter(order, [low, high], btype='bandpass')
        #b, a = signal.iirfilter(order, [low, high], rs=60, btype='band', analog = True, ftype =
'cheby2')
        y = signal.lfilter(b, a, data)

        return y

    def process_dataset(labels,train_audio_path):
        i = 1
        durations = []
        sum = 0
        processedAudioFiles = []
        all_label = []
        sr = 8000
        print("Signal processing")
        for label in labels:
            files = list(train_audio_path.glob(label + '/*.wav'))
            print("processing " + label)
            for file in files:
                originalSignal, sr = librosa.load(str(file), sr=8000, mono=True)
                processedSignal = bandpassIIRFilter(originalSignal, sr)
                processedSignal, index = librosa.effects.trim(processedSignal)
                duration = librosa.get_duration(processedSignal)
                sum+=duration
                processedAudioFiles.append(processedSignal)
                i+=1
                all_label.append(label)
                if sr != 8000:
                    print(sr)
```

```
    meanDuration=sum/i
    return processedAudioFiles,all_label,sr,meanDuration

def process_predict_dataset(labels,train_audio_path):
    i = 1
    durations = []
    sum = 0
    processedAudioFiles = []
    all_label = []
    sr = 8000
    print("Signal processing")
    for label in labels:
        files = list(train_audio_path.glob(label + '/*.wav'))
        print("processing " + label)
        for file in files:
            originalSignal, sr = librosa.load(str(file), sr=8000, mono=True)
            processedSignal = bandpassIIRFilter(originalSignal, sr)
            processedSignal, index = librosa.effects.trim(processedSignal)
            duration = librosa.get_duration(processedSignal)
            sum+=duration
            processedAudioFiles.append(processedSignal)
            i+=1
            all_label.append(label)
            if sr != 8000:
                print(sr)

    meanDuration=sum/i
    return processedAudioFiles,sr,meanDuration

def resample_dataset(labels,train_audio_path):
    i = 1
    durations = []
```

```
        sum = 0
        processedAudioFiles = []
        all_label = []
        sr = 8000
        print("Signal processing")
        for label in labels:
            files = list(train_audio_path.glob(label + '/*.wav'))
            print("processing " + label)
            for file in files:
                originalSignal, sr = librosa.load(str(file), sr=sr, mono=True)
                processedAudioFiles.append(originalSignal)
                i+=1
                all_label.append(label)
                if sr != 8000:
                    print(sr)

        meanDuration=sum/i
        return processedAudioFiles,all_label,sr,meanDuration

    # Neural here
    #x_tr, x_val, y_tr, y_val = train_test_split(np.array(x_), np.array(y_), stratify=y_,
test_size=0.2, random_state=777, shuffle=True)
```

● *split_to_words.py*

```
import os
import numpy as np
from pydub import AudioSegment
from pydub.silence import split_on_silence
import sounddevice as sd
import soundfile as sf
import matplotlib.pyplot as plt
import librosa   # for audio processing
import IPython.display as ipd
from keras.utils import np_utils
from keras.layers import Dense, Dropout, Flatten, Conv1D, Input, MaxPooling1D
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import backend as K
from keras.models import load_model


########## SPLIT THE PHRASE INTO WORDS #############

def splitter(filename,directory):
    sound_file = AudioSegment.from_wav(filename)
    audio_chunks = split_on_silence(sound_file,
                                    # must be silent for at least half a second
                                    min_silence_len=200,

                                    # consider it silent if quieter than -64 dBFS
                                    silence_thresh=-64
                                    )

    # Export each digit of a sequence in a .wav file.
    for i, chunk in enumerate(audio_chunks):
        out_file = directory+"/chunk{0}.wav".format(i)
```

```
print("exporting", out_file)
chunk.export(out_file, format="wav")
```

● *train_model.py*

```
import numpy as np
import os
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from keras.utils import np_utils, to_categorical
from keras.layers import Dense, Dropout, Flatten, Conv1D, Input, MaxPooling1D,
SimpleRNN
from keras.callbacks import EarlyStopping, ModelCheckpoint, History
from keras.models import load_model, Sequential
from keras.models import Model
from keras import backend as K

import matplotlib.pyplot as plt

def train_dense(all_features, all_label, labels, model_name="model_dense"):
    current_directory = os.getcwd()
    all_features = np.array(all_features)
    all_label = np.array(all_label)
    print('Size of X:\n', all_features.shape)
    print('Size of Y:\n', all_label.shape)

    # sc = StandardScaler()
    # x_ = sc.fit_transform(all_features)
    x_ = all_features
    y_ = all_label
    le = LabelEncoder()
    y_ = le.fit_transform(all_label)
    classes = list(le.classes_)
    np.save(current_directory + '/classes_dense.npy', le.classes_)
    y_ = np_utils.to_categorical(y_, num_classes=len(labels))  # From int to one-hot
```

```
        print('to_categorical:\n', y_)
        x_tr, x_val, y_tr, y_val = train_test_split(np.array(x_), np.array(y_), stratify=y_,
test_size=0.2,
                                                    random_state=777, shuffle=True)
        print('x_tr: ', x_tr.shape, '\nx_val: ', x_val.shape, '\ny_tr: ', y_tr.shape, '\ny_val: ',
y_val.shape)
        n_cols = x_tr.shape[1]

        model = Sequential()

        model.add(Dense(36, activation='relu', input_dim=n_cols))  # ,
activity_regularizer=l1(0.01)))
        model.add(Dense(52, activation='relu'))
        model.add(Dense(52, activation='relu'))
        model.add(Dense(36, activation='relu'))
        model.add(Dense(24, activation='relu'))

        model.add(Dense(len(labels), activation='softmax', name='pred'))

        # Model Summary
        model.summary()

        # Compile Model
        model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])
        # Early Stopping
        es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10,
min_delta=0.05)
        # Keep only a single checkpoint, the best over test accuracy.
        filepath = current_directory + '/dense_' + model_name + '.hdf5'
        mc = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
        model.save(filepath, True, True)
```

```
        # Fit Model
        history = model.fit(x_tr, y_tr, epochs=100, callbacks=[es, mc], batch_size=64,
validation_data=(x_val, y_val))

        plt.plot(history.history['loss'], label='train')
        plt.plot(history.history['val_loss'], label='test')
        plt.legend()
        plt.show()

        # evaluate model
        _, accuracy = model.evaluate(x_tr, y_tr)
        print('Accuracy: %.2f' % (accuracy * 100))
        # print(y_val)
        y_pred = model.predict_classes(x_tr)

        # Scores
        score_train = model.evaluate(x_tr, y_tr, verbose=0)
        score_val = model.evaluate(x_val, y_val, verbose=0)

        print('Train Score: ', score_train, '\nValidation Score: ', score_val)

        return filepath


    def train_convolutional(all_features,all_label,labels, model_name="model_conv"):
        all_features = np.array(all_features)
        current_directory=os.getcwd()

        # Neural Network Classification
        # Label Encoding and train-test split
        le = LabelEncoder()  # Encode labels from (zero, one, ...) to (0, ..., 1)
        y = le.fit_transform(all_label)
```

```
    classes = list(le.classes_)
    np.save( current_directory + '/classes_conv.npy', le.classes_)
    y = np_utils.to_categorical(y, num_classes=len(labels))  # From int to one-hot
    all_features = np.array(all_features).reshape(-1, len(all_features[0]), 1)  # only for convolutional nn (2D to 3D)
    x_tr, x_val, y_tr, y_val = train_test_split(np.array(all_features), np.array(y), stratify=y, test_size=0.2,
                                                random_state=777, shuffle=True)

    K.clear_session()

    print(all_features[0].shape, len(all_features[0]))

    inputs = Input(shape=(len(all_features[0]), 1))

    # First Conv1D layer
    conv = Conv1D(8, 13, padding='valid', activation='relu', strides=1)(inputs)
    conv = MaxPooling1D(3)(conv)
    conv = Dropout(0.3)(conv)

    # Second Conv1D layer
    conv = Conv1D(16, 11, padding='valid', activation='relu', strides=1)(conv)
    conv = MaxPooling1D(3)(conv)
    conv = Dropout(0.3)(conv)

    # Third Conv1D layer
    conv = Conv1D(32, 9, padding='valid', activation='relu', strides=1)(conv)
    conv = MaxPooling1D(3)(conv)
    conv = Dropout(0.3)(conv)

    # Fourth Conv1D layer
    conv = Conv1D(64, 7, padding='valid', activation='relu', strides=1)(conv)
    conv = MaxPooling1D(3)(conv)
```

```
        conv = Dropout(0.3)(conv)

        # Flatten layer
        conv = Flatten()(conv)

        # Dense Layer 1
        conv = Dense(256, activation='relu')(conv)
        conv = Dropout(0.3)(conv)

        # Dense Layer 2
        conv = Dense(128, activation='relu')(conv)
        conv = Dropout(0.3)(conv)

        outputs = Dense(len(labels), activation='softmax')(conv)

        model = Model(inputs, outputs)
        model.summary()

        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
        es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10,
min_delta=0.0001)

        filepath = current_directory + '/conv_' + model_name + '.hdf5'

        # Keep only a single checkpoint, the best over test accuracy.
        mc = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
        model.save(filepath, True, True)

        history = model.fit(x_tr, y_tr, epochs=100, callbacks=[es, mc], batch_size=32,
validation_data=(x_val, y_val))
```

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

score_train = model.evaluate(x_tr, y_tr, verbose=0)
score_val = model.evaluate(x_val, y_val, verbose=0)

print('Train Score: ', score_train, '\nValidation Score: ', score_val)

return filepath
```