



Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών
Υπολογιστών

2η Εργαστηρική Άσκηση

Ανγγνώριση Προτύπων

Αλεξανδρόπουλος Σταμάτης

03117060

el17060@mail.ntua.com

Γκότση Πολυτίμη Άννα

03117201

el17201@mail.ntua.com

Περιεχόμενα

Εισαγωγή	1
Βήμα 1	2
Βήμα 2	3
Βήμα 3	4
Βήμα 4	13
Βήμα 5	14
Βήμα 6	16
Βήμα 7	20
Βήμα 8	23
Βήμα 9	26
Βήμα 10	27
Βήμα 11	27
Βήμα 12	29
Βήμα 13	31
Βήμα 14	43

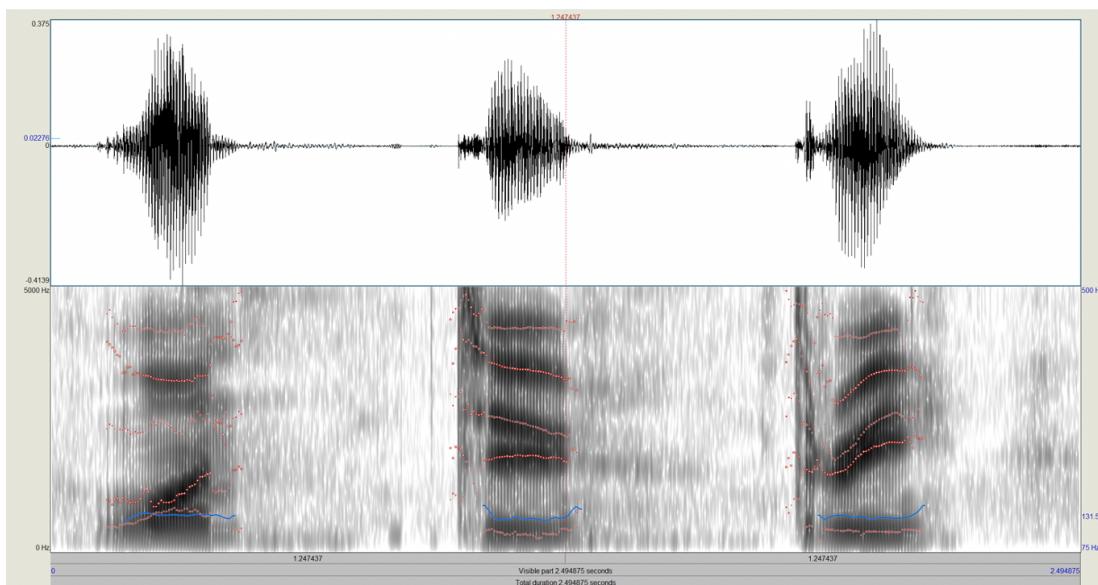
Εισαγωγή

Στόχος αυτής της εργαστηριακής άσκησης είναι η επεξεργασία και αναγνώριση σημάτων φωνής με τη χρήση κατάλληλων εργαλείων της Python. Αρχικά πραγματοποιείται η μελέτη τους και ανάλυση βασικών χαρακτηριστικών τους στο πεδίο της συχνότητας μέσω της μελέτης της κυματομορφής και του spectrogram τους με το πρόγραμμα Praat. Στην συνέχεια γίνεται εξαγωγή χαρακτηριστικών από σήματα φωνής, μελέτη των εξαγόμενων χαρακτηριστικών και χρήση τους για αναγνώριση φωνής καθώς και μια εισαγωγική μελέτη των RNN νευρωνικών δικτύων.

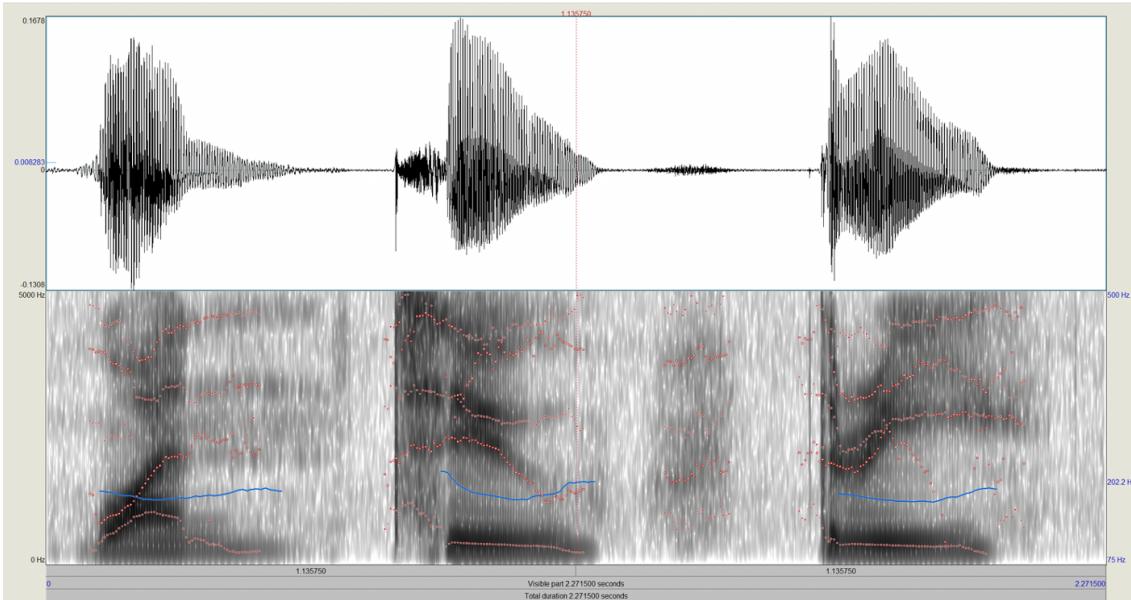
Βήμα 1

Αρχικά πραγματοποιούμε επεξεργασία των αρχείων onetwothree1.wav και onetwothree8.wav με χρήση του προγράμματος Praat. Τα αρχεία αυτά περιλαμβάνουν την ηχητική προσφώνηση της πρότασης “one two three” από έναν άνδρα και από μια γυναίκα ομιλητή αντίστοιχα. Μετά την εισαγωγή των ηχητικών αρχείων στο σύστημα Praat παρατηρούμε την κυματομορφή και το spectrogram κάθε ηχητικού σήματος:

- Για τον άνδρα ομιλητή (onetwothree1.wav)



- Για την γυναίκα ομιλητή (onetwothree8.wav)



Παρατηρούμε ότι το σήμα από την γυναίκα ομιλητή έχει μεγαλύτερο πλάτος και πιο μεγάλες συχνότητες.

Στην συνέχεια εξάγουμε τη μέση τιμή του pitch και τη μέση συχνότητα πρώτου (F1), δεύτερου(F2) και τρίτου(F3) Formant στα φωνήντα “α”, “ου”, “ι” για τα 3 φηφία και για κάθε ομιλητή. Τα φωνήματα αυτά είναι περιοδικά και επομένως μπορούμε να τα βρούμε είτε ακούγοντας το ηχητικό αρχείο είτε βλέποντας που γίνεται περιοδική η κυματομορφή είτε συνδυάζοντας και τις δύο μεθόδους. Έχουμε τα εξής αποτελέσματα:

Ομιλητής	Φωνήν	Χρονική Διάρκεια (sec)	Μέση Τιμή Pitch (Hz)	Formant 1(Hz)	Formant 2(Hz)	Formant 3(Hz)
άντρας	α	0.299 - 0388	134.675	757.513	1249.326	2413.343
γυναίκα	α	0.167 - 0.285	177.849	880.211	1571.909	3004.935
άντρας	ου	1.056 - 1.273	130.710	361.707	1785.622	2388.273
γυναίκα	ου	0.861 - 1.184	188.658	347.835	1722.251	2686.739
άντρας	ι	1.929 - 2.126	132.670	399.743	1984.117	2423.754
γυναίκα	ι	1.741 - 2.034	179.282	396.339	2178.810	1858.908

Από τα παραπάνω μπορούμε να δούμε πως το pitch εξαρτάται από τον ομιλητή. Είναι δηλαδή πολύ κοντά σε τιμές για το κάθε φωνήν. Έχουμε, λοιπόν:

- Για τον άντρα ομιλητή: pitch mean $\approx 132\text{Hz}$
- Για την γυναίκα ομιλητή: pitch mean $\approx 181\text{Hz}$

Γενικά παρατηρούμε ότι τα formants επηρεάζονται μεν από τον ομιλητή (κυρίως από το αν είναι άντρας ή γυναίκα αλλά όχι μόνο), εξαρτώνται όμως κυρίως από το φωνήν (παρατηρούμε ότι τα formants για το ίδιο φωνήν είναι παρόμοια και για τους δύο ομιλητές, αλλά τα formants για διαφορετικό φωνήν διαφέρουν σημαντικά, διιδίτερα αυτά για το φωνήν “α”).

Παρατηρούμε επίσης ότι τα formants του φωνήντος “α” διαφέρουν αρκετά από αυτά των “ι” και “ου”, ενώ τα formants των δύο τελευταίων φωνήντων είναι πιο κοντά. Αυτό, όπως διαπιστώνουμε από σχετικές μελέτες των formants στην βιβλιογραφία ήταν αναμενόμενο.

Τέλος, αξίζει να σημειώσουμε πως, κανόνις η επιλογή των τμημάτων του ηχητικού αρχείου που αποτελούν το κάθε φωνήν σε μια λέξη έγινε χειροκίνητα, τα μετρούμενα pitch και formants μπορεί να προκύπτουν ελαφρώς διαφορετικά σε διαφορετικές μετρήσεις.

Βήμα 2

Στο βήμα αυτό κατασκευάζουμε μία συνάρτηση (data parser) που διαβάζει όλα τα αρχεία ήχου που δίνονται στο φάκελο digits/. Συνολικά μας δίνονται 133 αρχεία ήχου. Στο dataset των φηφίων μας έχουμε 15 ομιλητές για τα φηφία 1-9. Όμως όπως μπορούμε να παρατηρήσουμε, δεν έχουμε στα δεδομένα μας τα αρχεία six12.wav και eight7.wav γεγονός που δικαιολογεί την ύπαρξη 133 αρχείων ήχου και όχι 135.

Η συνάρτηση (data parser) επιστρέφει τρεις λίστες με πλήρη αντιστοίχιση των στοιχείων τους. Η πρώτη λίστα (wavs) περιέχει τα ηχητικά αρχεία όπως αποθηκεύτηκαν μέσω διαδικασιών της librosa, που είναι βιβλιοθήκη dsp της python, η δεύτερη (speakers) περιέχει το id του ομιλητή και η τρίτη (digits) το φηφίο που εκφωνεί ο ομιλητής.

Τονίζουμε ότι για ευκολία στους υπολογισμούς που θα ακολουθήσουν στην συνέχεια, κάνουμε ταρ μη τα φηφία ('one', 'two', 'three', ...) της λίστας digits σε αριθμούς (1, 2, 3, ...)

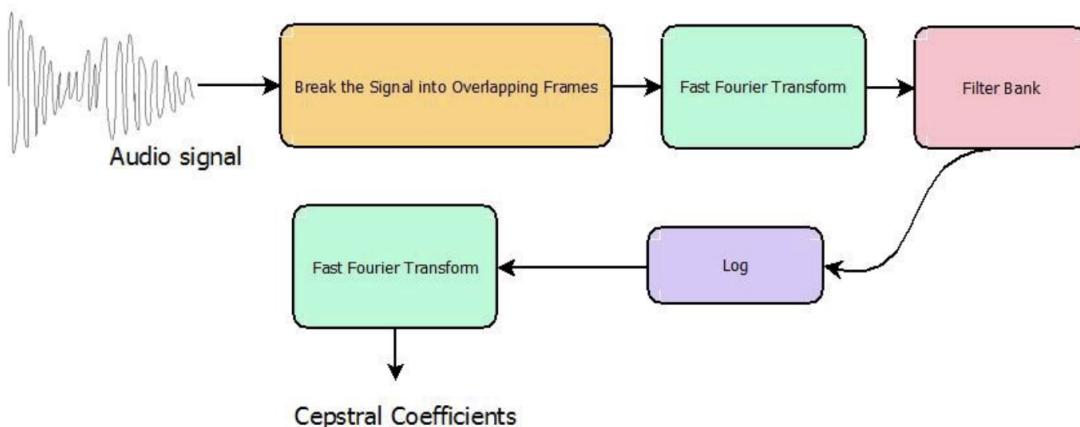
Βήμα 3

Στο βήμα αυτό πραγματοποιούμε feature extraction στα σήματα της φωνής. Συγκεκριμένα εξάγουμε με χρήση της βιβλιοθήκης librosa τα Mel-Frequency Cepstral Coefficients (MFCCs) για κάθε αρχείο ήχου.

Τα Mel-Frequency Cepstral Coefficients χαρακτηριστικά, προσπαθούν να περιγράψουν την ανθρώπινη αντίληψη για την συχνότητα του ήχου. Αυτά βασίζονται στο φάσμα του σήματος και υπολογίζονται με τη βοήθεια του μετασχηματισμού Fourier μικρού χρόνου (STFT). Ο τρόπος υπολογισμού τους είναι ο εξής:

1. Με τη χρήση ενός παραθύρου διαμερίζεται το σήμα σε πλαίσια
2. Υπολογίζεται το φάσμα με χρήση του γρήγορου μετασχηματισμού Fourier (FFT)
3. Μετατρέπονται τα αποτελέσματα του γρήγορου μετασχηματισμού Fourier μέσω μιας συστοιχίας φίλτρων σε νέα μορφή
4. Υπολογίζεται ο λογάριθμος αυτών με βάση το 10
5. Υπολογίζεται ο μετασχηματισμός συνημίτονου

Ουσιαστικά αυτό που κάνει μοναδικά τα Mel-Frequency Cepstral Coefficients χαρακτηριστικά είναι η συστοιχία των φίλτρων. Αυτά τα χαρακτηριστικά χρησιμοποιούνται με μεγάλη επιτυχία τόσο στην αναγνώριση ομιλίας όσο και στην αναγνώριση των ειδών της μουσικής, αφού περιγράφουν καλύτερα από άλλες τεχνικές την διαισθητική αντίληψη του φάσματος. Σχηματικά έχουμε:



Για την short time frequency analysis χρησιμοποιούμε κυλιόμενο παράθυρο μήκους 25ms και βήμα

10ms. Αυτό σημαίνει πως υπάρχει σημαντική επικάλυψη μεταξύ των παραθύρων, κάτιο το οποίο εξασφαλίζει μεγαλύτερη ομαλότητα στις μεταβάσεις μεταξύ αυτών. Έτσι ορίζονται το μήκος παραθύρων και το βήμα μεταξύ παραθύρων χρησιμοποιούμε την συνάρτηση librosa.feature.mfcc που εξάγει τις MFCCs coefficients, ορίζοντας να επιστρέψει 13 coefficients. Ο αριθμός 13 είναι το σύνηθες πλήρος MFCCs που εξάγουμε και έχει προκύψει από διάφορες μελέτες. Το αποτέλεσμα είναι αποθηκευμένο στη λίστα frames, η οποία επομένως περιλαμβάνει τα 13 MFCCs coefficients για κάθε παράθυρο, για καθένα από τα αρχεία ήχου.

Στην συνέχεια, σύριγνωνα με τα ζητούμενα, εξάγουμε τις πρώτες και τις δεύτερες τοπικές παραγώγους (deltas, delta-deltas) των χαρακτηριστικών MFCCs μέσω της συνάρτησης librosa.feature.delta. Αποθηκεύουμε τα αποτελέσματα στις λίστες deltas1,deltas2.

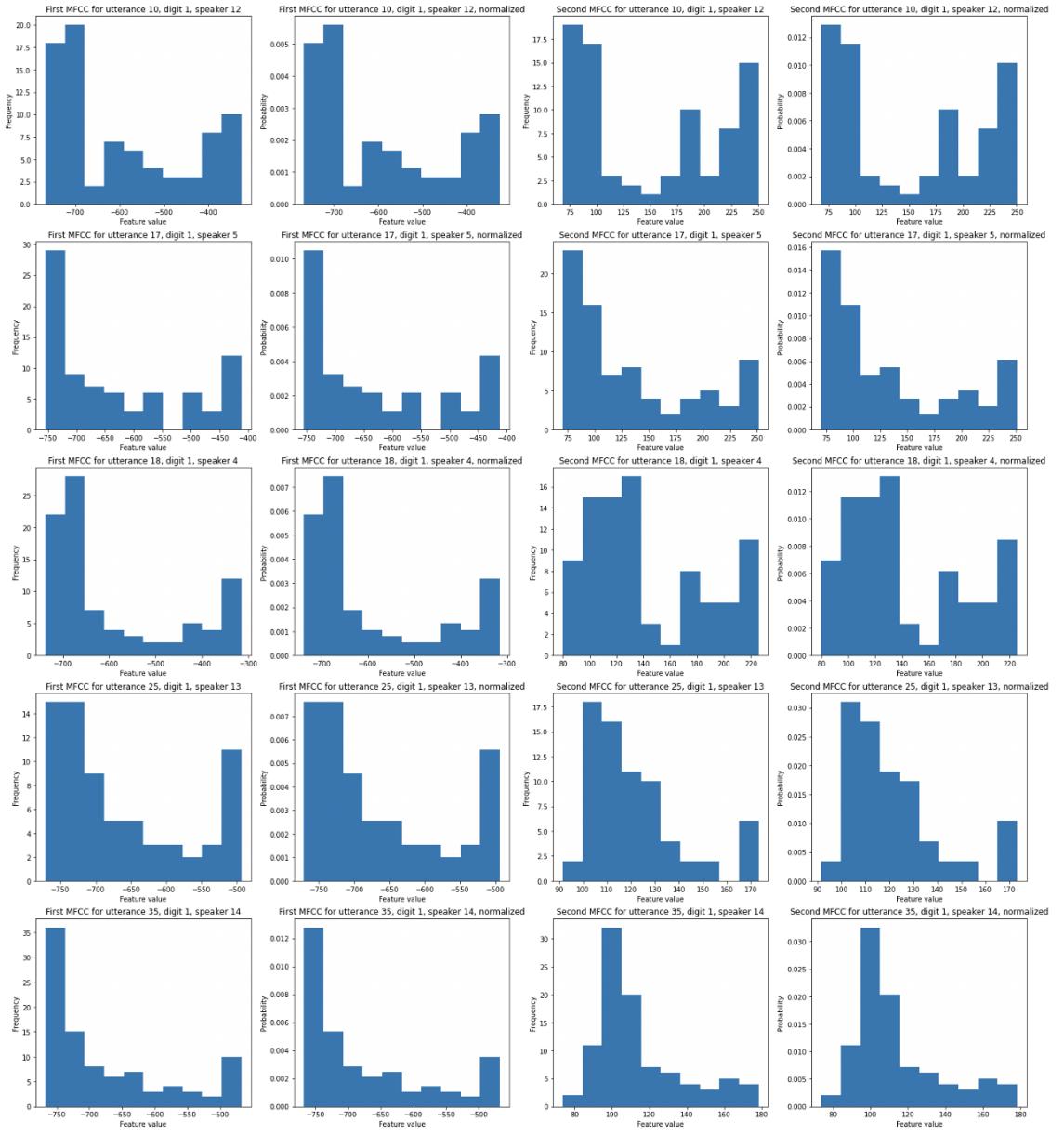
Βήμα 4

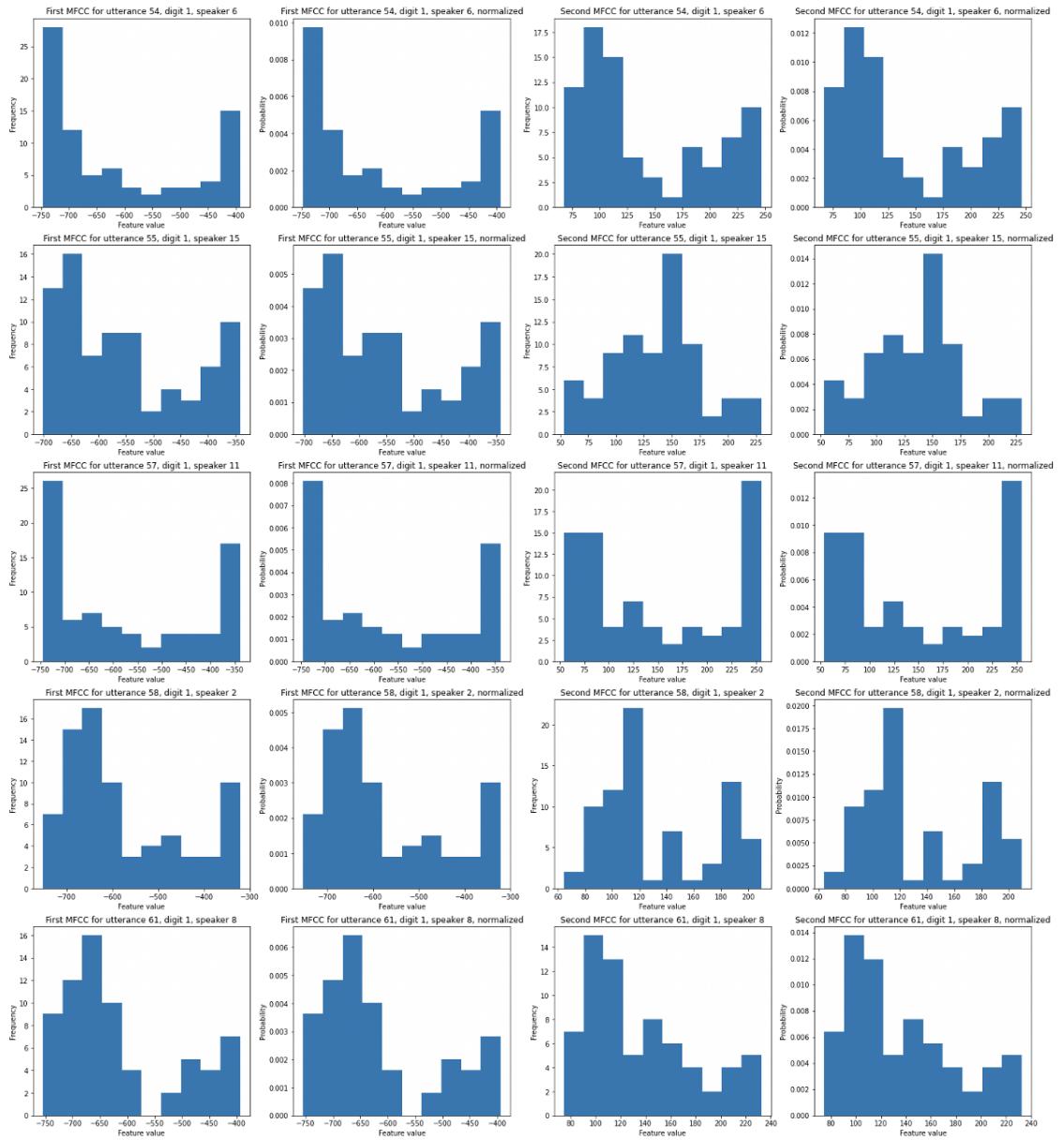
Στο βήμα αυτό αναπαριστούμε τα ιστογράμματα του 1ου και του 2ου MFCC των ψηφίων 6 και 1 για όλες τους τις εκφωνήσεις. Τα ψηφία αυτά προέκυψαν με βάση τη δοθείσα σημείωση. Συγκεκριμένα έχουμε:

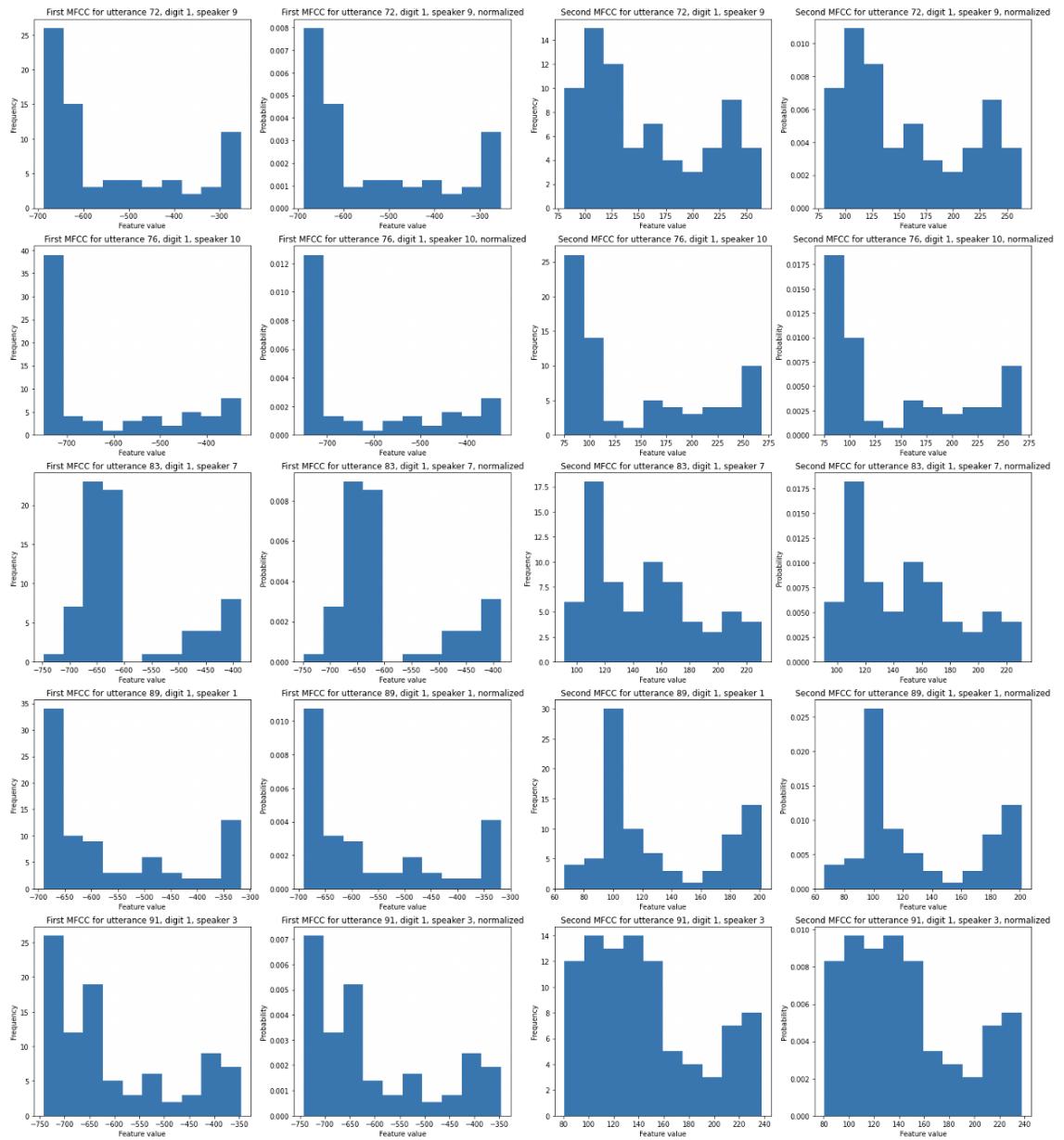
- 03117060 → 6
- 03117201 → 1

Επισημαίνεται ότι για μεγαλύτερη ευκολία στη σύγκριση των διαγραμμάτων, παραθέτουμε και τα αποτελέσματα με κανονικοποιημένο τον άξονα y. Έχουμε τα εξής αποτελέσματα:

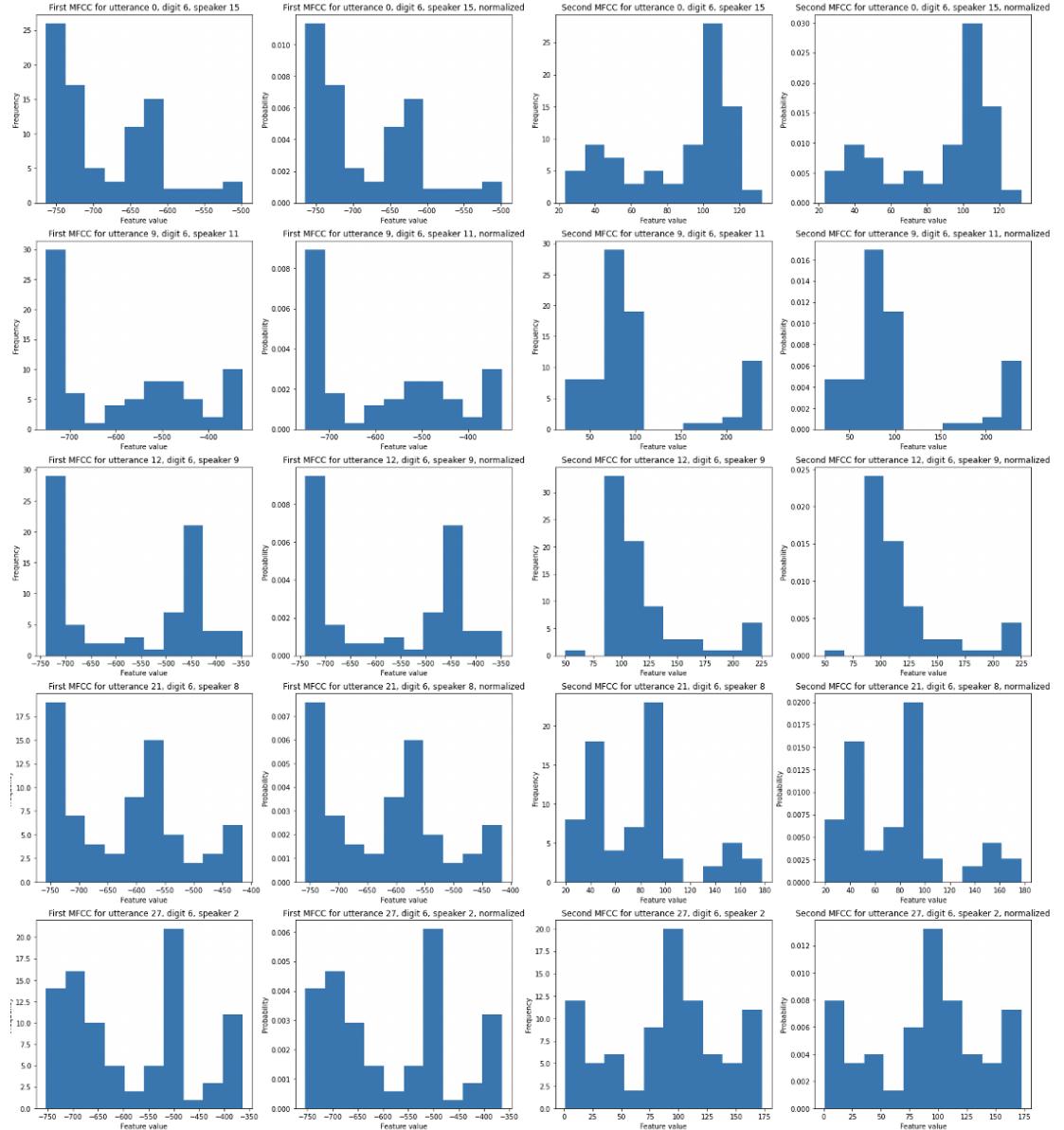
- Για το ψηφίο 1 το ιστόγραμμα (κανονικοποιημένο και μη) της πρώτης και δεύτερης coefficient των MFCCs για όλους τους ομιλητές είναι:

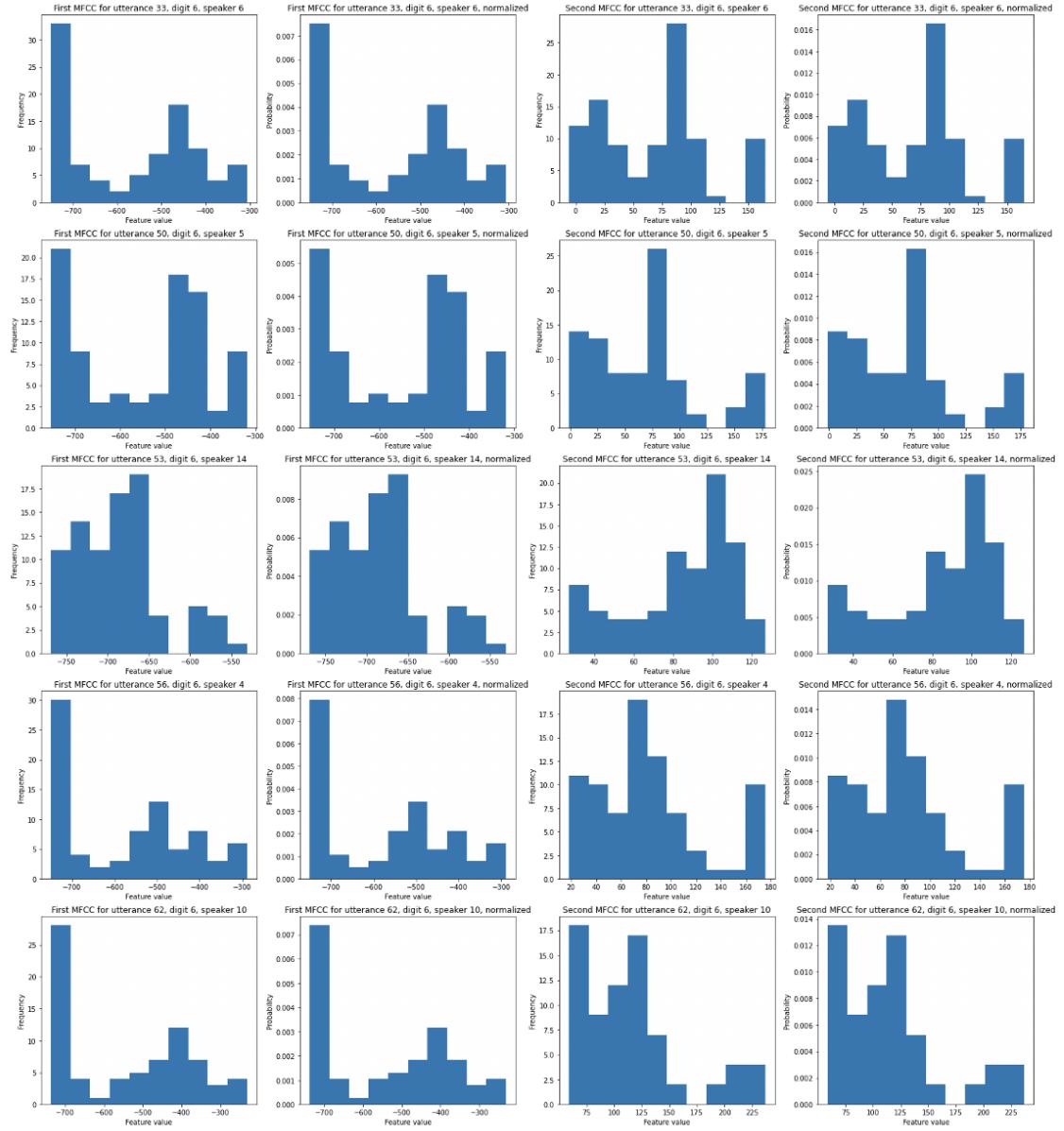


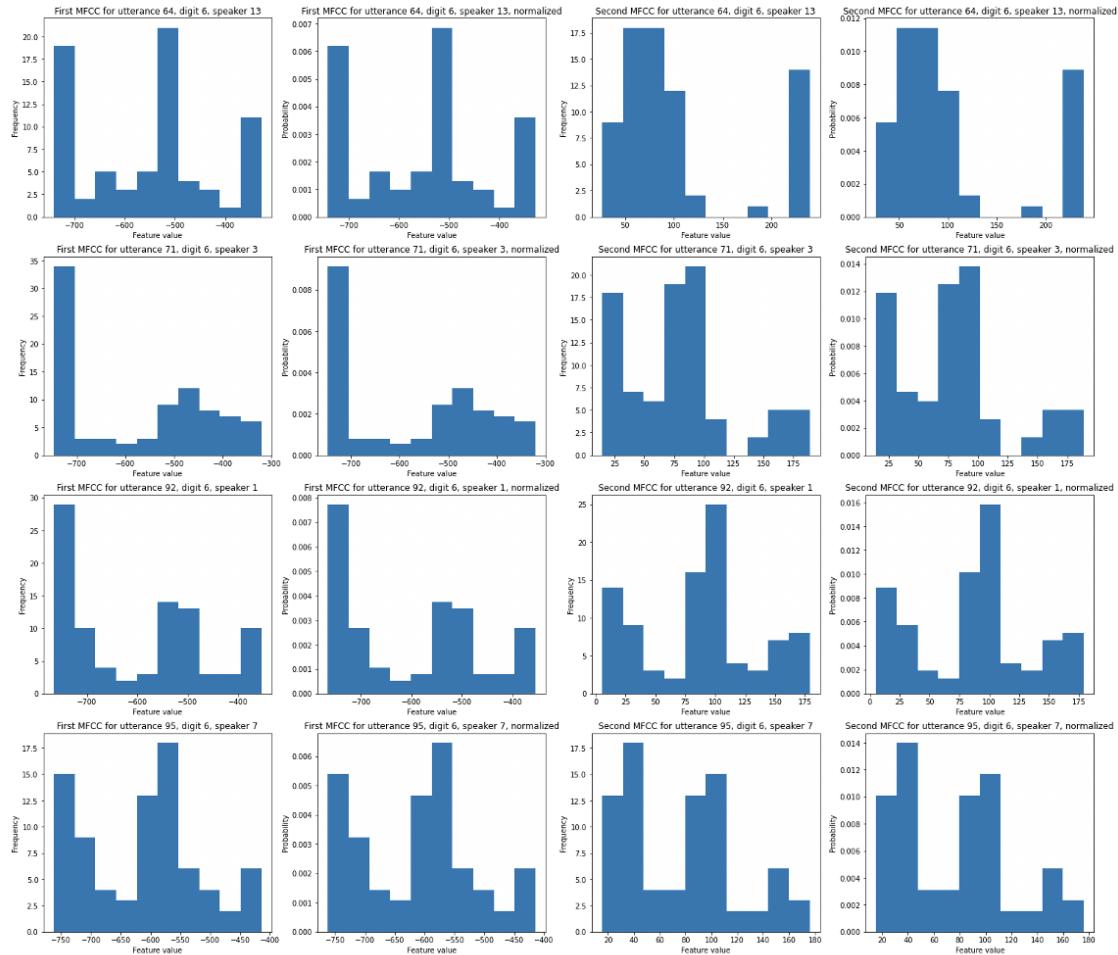




- Για το ψηφίο 6 το ιστόγραμμα (χανονικοποιημένο και μη) της πρώτης και δεύτερης coefficient των MFCCs για όλους τους ομιλητές είναι:







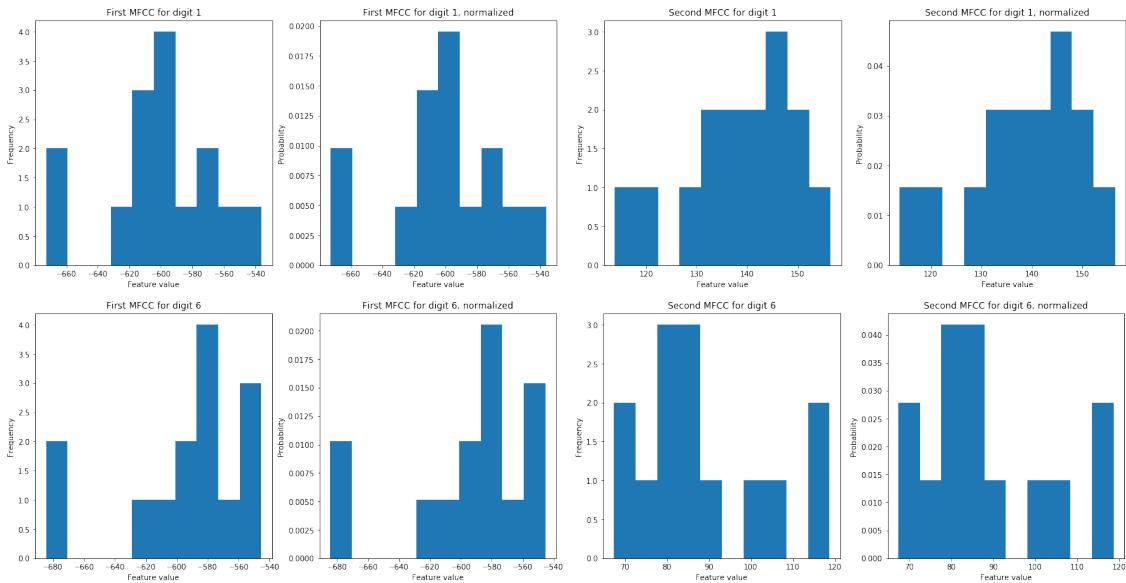
Όπως αναφέραμε και παραπάνω λείπει από το dataset το αρχείο six12.wav γιαυτό και δεν έχουμε διαγράμματα για το ομιλητή 12.

Συγχρίνοντας τα ιστογράμματα της πρώτης coefficient των MFCCs με της δεύτερης coefficient βλέπουμε ότι παρουσιάζουν διαφορές αρχικά ως προς το σχήμα του ιστογράμματος αλλά και ως προς τις τιμές καιώς η πρώτη MFCC κυμαίνεται σε αρνητικές τιμές και συγκεκριμένα μεταξύ [-800, -350] ενώ η δεύτερη σε θετικές τιμές και συγκεκριμένα μεταξύ [0, 300].

Επιλέον, συγχρίνοντας το ιστόγραμμα της ίδιας MFCC για το ίδιο ψηφίο αλλά διαφορετικούς ομιλητές βλέπουμε πως αυτό έχει αντίστοιχο εύρος τιμών και ορισμένες ομοιότητες στην γενική του μορφή, ενώ όμως αρκετά διαφορετικό ανά ομιλητή.

Από την άλλη συγχρίνοντας την ίδια coefficient των MFCCs για διαφορετικά ψηφία παρατηρούμε ότι εμφανίζουν κάποιες ομοιότητες τόσο στο σχήμα όσο και στο έυρος των ιστογραμμάτων.

Παρακάτω παραθέτουμε και τα ιστόγραμμα για το ψηφίο 1 και 6 αντίστοιχα και το μέσο MFCC για όλους τους ομιλητές:



Παρατηρούμε ότι τα ιστογράμματα για κάθε ψηφίο διαφέρουν μεταξύ τους, οπότε είναι δυνατόν να ξεχωρίσουμε το κάθε ψηφίο.

Επομένως, μπορούμε να συμπεράνουμε ότι οι MFCCs ναι μεν είναι αρκετά διαφοροποιημένες μεταξύ τους αλλα μπορούμε να εξάγουμε πληροφορίες συγκρίνοντας την ίδια συνιστώσα για διαφορετικά ψηφία.

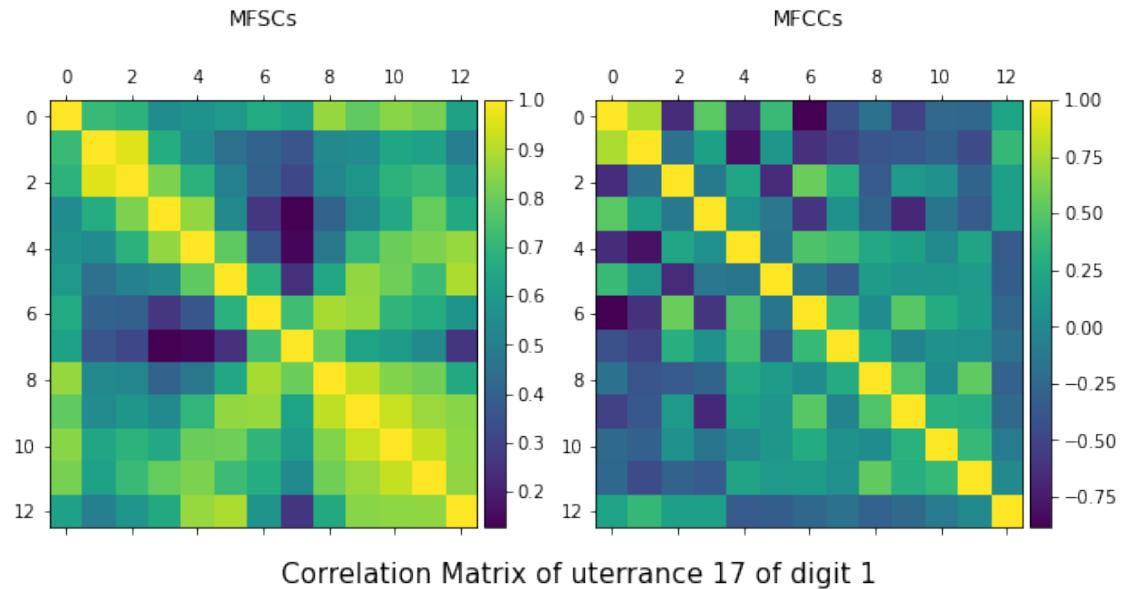
Συνεχίζοντας στο δεύτερο υποερώτημα του βήματος αυτού, δηλαδή την εξαγωγή MFSCs, αξίζει να σχολιάσουμε το τι είναι η MEL χλίμακα. Ουσιαστικά σχετίζεται με τον τρόπο που ο άνθρωπος αντιλαμβάνεται τις διαφορετικές συχνότητες και έτσι βλέπουμε ότι είναι σχεδόν γραμμική για κάτω των 1000 Hz μετά γίνεται λογαριθμική.

Στο υποερώτημα αυτό εξάγουμε για 2 εκφωνήσεις των ψηφίων 6 και 1 από 2 διαφορετικούς ομιλητές τα Mel Filterbank Spectral Coefficients (MFSCs) , δηλαδή τα χαρακτηριστικά που εξάγονται αφού εφαρμοστεί η συστοιχία φίλτρων της χλίμακας Mel πάνω στο φάσμα του σήματος φωνής αλλά χωρίς να εφαρμοστεί στο τέλος ο μετασχηματισμός DCT. Γνωρίζουμε από την θεωρία ότι σε αυτόν τον χώρο τα χαρακτηριστικά είναι πιο συσχετισμένα από ότι στο cepstrum, γεγονός που αποτελεί και τον λόγο για τον οποίο χρησιμοποιούμε συνήθως το cepstrum, γιατί μας δίνει αυτή την ανεξαρτησία.

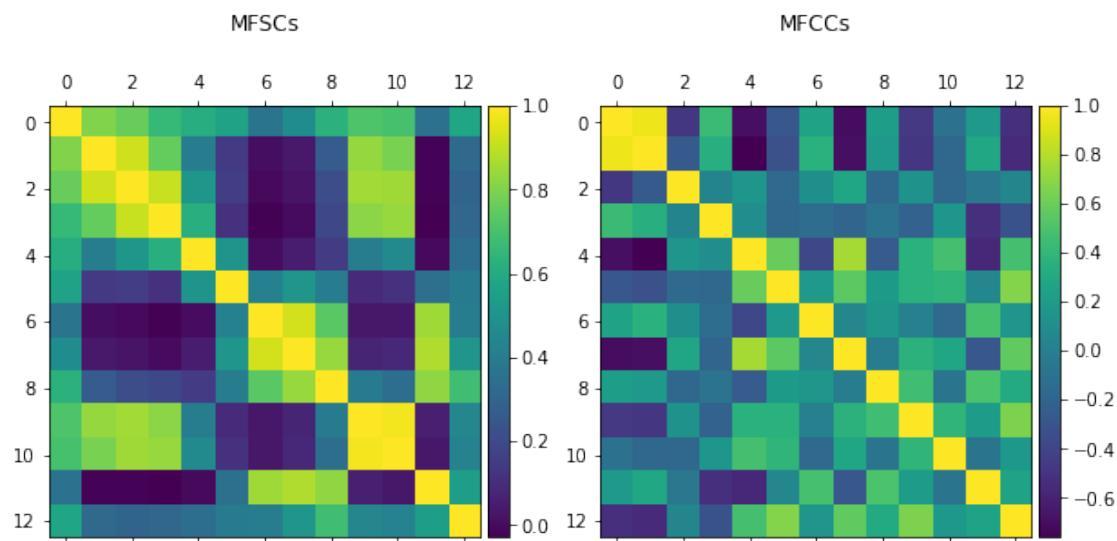
Μέσω της συνάρτησης librosa.feature.melspectrogram τυπώνουμε τους πίνακες συσχέτισης των MFSCs και των MFCCs για τις δύο εκφωνήσεις για τα ψηφία 1 και 6. Έτσι παίρνουμε τα ακόλουθα 8 διαγράμματα (4 για κάθε ψηφίο):

Αρχικά βλέπουμε τα 4 διαγράμματα που αφορούν το ψηφίο 1:

Correlation Matrix of utterance 35 of digit 1

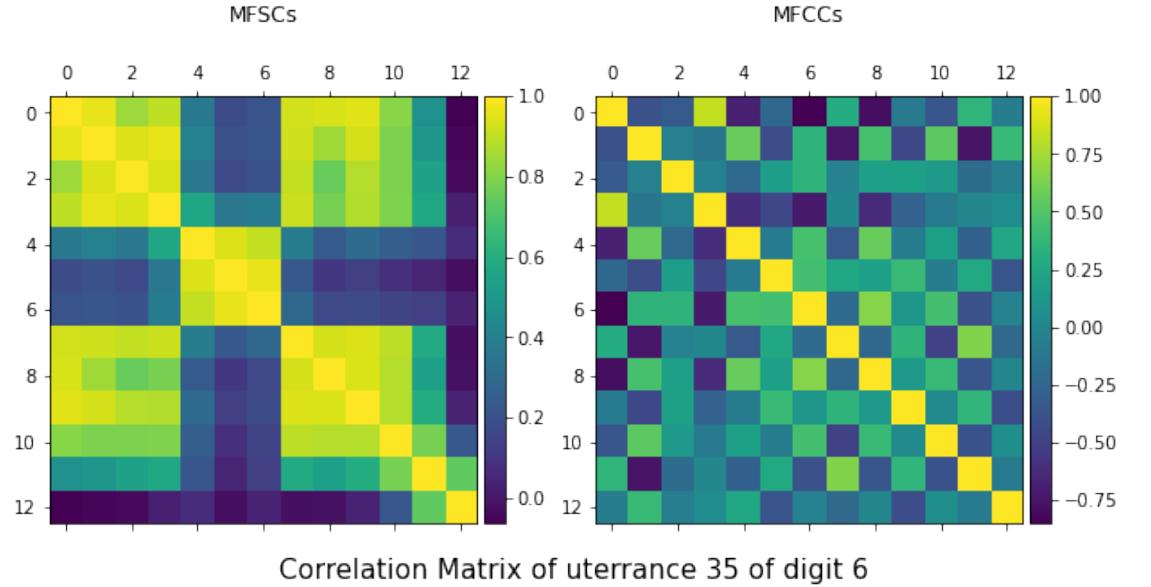


Correlation Matrix of utterance 17 of digit 1

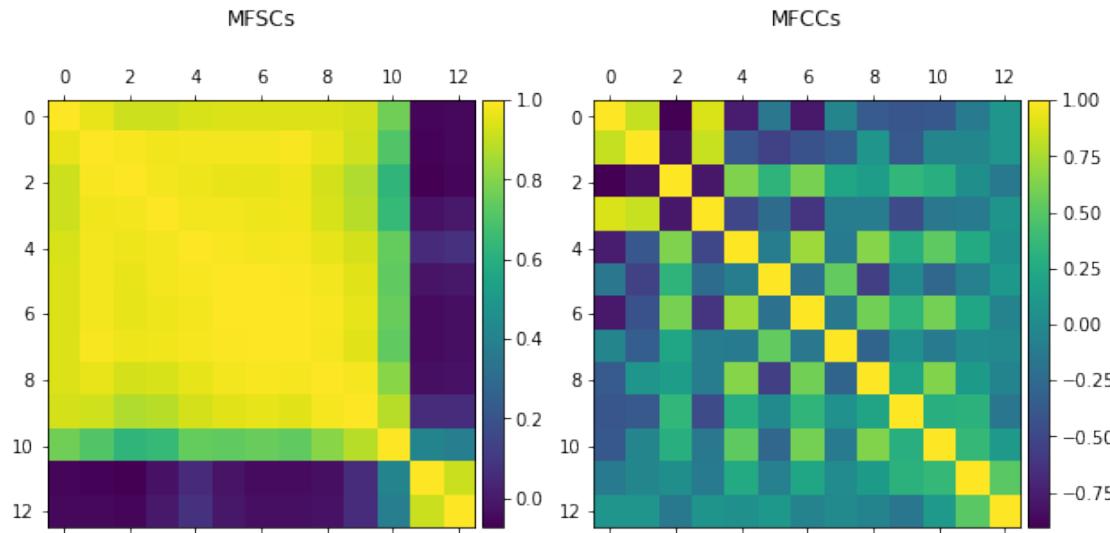


Έπειτα βλέπουμε τα 4 διαγράμματα που αφορούν το ψηφίο 6:

Correlation Matrix of uterrance 0 of digit 6



Correlation Matrix of uterrance 35 of digit 6



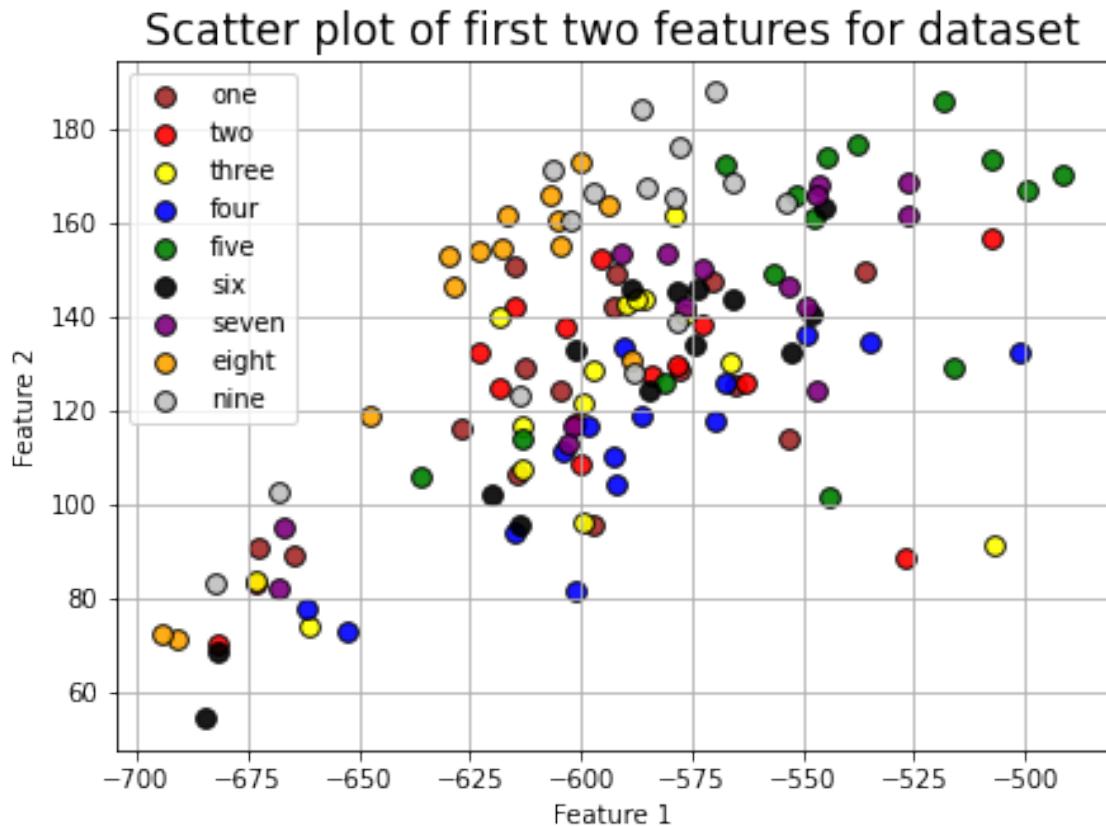
Παρατηρούμε ότι τα χαρακτηριστικά MFSCs είναι πολύ πιο έντονα συσχετισμένα, ενώ τα χαρακτηριστικά MFCCs εμφανίζουν πολύ μικρότερες συσχετίσεις. Συγκεκριμένα στα MFCCs μπορούμε να δούμε ότι στον πίνακα συσχετίσης έχουμε παντού 1 στην κύρια διαγώνιο για όλα τα στοιχεία και αρκετά χαμηλότερες συσχετίσεις στις περισσότερες άλλες θέσεις. Από την άλλη, ιδιαίτερα για το ψηφίο 6, τα MFSCs μπορούν να ξεχωρίσουν δύο clusters με τα στοιχεία με συσχέτιση 1 και αυτά που έχουν 0. Αυτό είναι λιγότερο έντονο στην περίπτωση του ψηφίου 1, αλλά και εκεί υπάρχουν ομάδες στοιχείων με πολύ μεγάλη συσχέτιση και άλλες με πολύ μικρή.

Γενικά, θέλουμε να έχουμε ασυσχέτιστα χαρακτηριστικά, καθώς αυτά προσθέτουν περισσότερη νέα πληροφορία στο σύστημα. Επιπρόσθετα για ασυσχέτιστα δεδομένα μπορούμε να χρησιμοποιήσουμε διαγώνιο πίνακα συνδιακύμανσης, γεγονός που παρέχει υπολογιστική ευκολία δεδομένου ότι ο διαγώνιος πίνακας αντιστρέφεται πολύ πιο εύκολα. Συμβάλλει, επίσης και στον περιορισμό του ο-ερφιττινγ, μειώνοντας το πλήθος των παραμέτρων του μοντέλου. Σημειώνουμε τέλος ότι, για τα νευρωνικά μοντέλα, όπου η συσχέτιση των δεδομένων δεν είναι τόσο μεγάλο εμπόδιο, μπορούμε να επιλέξουμε να δουλέψουμε με τα MFCCs.

Βήμα 5

Στο βήμα αυτό υλοποιούμε μια πρώτη προσέγγιση για την αναγνώριση των ψηφίων μέσω της εξαγωγής ενός μοναδικού διανύσματος χαρακτηριστικών για κάθε εκφώνηση. Σε αυτό το σημείο ενώνουμε τις λίστες mfccs – deltas – delta-deltas και έπειτα για κάθε εκφώνηση δημιουργούμε ένα διάνυσμα παίρνοντας τη μέση τιμή και την τυπική απόκλιση κάθε χαρακτηριστικού για όλα τα παράθυρα της εκφώνησης. Τα παραπάνω υλοποιούνται μέσω της συνάρτησης calculate_unique_vector που βρίσκεται στο αρχείο κώδικα. Στη συνάρτηση αυτή αρχικά υπολογίζουμε τις μέσες τιμές και τις αποκλίσεις για κάθε feature και για όλα τα δείγματά μας σε κάθε ένα από τα mfccs – deltas - delta-deltas και έπειτα ενώνουμε σε έναν πίνακα όλα τα στοιχεία που βρήκαμε πριν. Έτσι, για κάθε δείγμα προκύπτουν 6 x 13=78 τιμές, αφού υπολογίσαμε το mean value και το std value για καθένα από τα mfccs – deltas - delta-deltas) και έχουμε 13 χαρακτηριστικά για καθένα από τα mfccs – deltas - delta-deltas . Αυτό έχει σαν αποτέλεσμα να προκύψει ένα πίνακας 133 x 78.

Στην συνέχεια απεικονίζουμε τις πρώτες δύο διαστάσεις του νέου διανύσματος (την μέση τιμή και την τυπική απόκλιση της πρώτης MFCC):



Από το παραπάνω διάγραμμα βλέπουμε ότι δε είναι εύκολο να ξεχωρίσουμε το ψηφίο σε σχέση με το που βρίσκεται στο επίπεδο, καθώς οι περιοχές των διαφορετικών κατηγοριών είναι επικαλυπτόμενες.

Βήμα 6

Στην συνέχεια υποβιβάζουμε τα δεδομένα σε χώρους μικρότερης διάστασης για να λάβουμε μία εικόνα του του συνόλου των χαρακτηριστικών του διανύσματος και όχι μόνο των δύο πρώτων συνιστώσων. Για τον σκοπό αυτό χρησιμοποιούμε την μέθοδο ανάλυσης σε πρωτεύοντες συνιστώσες (PCA) και παρουσιάζουμε τα δεδομένα μας σε χώρους 2 διαστάσεων και 3 διαστάσεων, μελετώντας ταυτόχρονα το ποσοστό απώλειας πληροφορίας λόγο της μείωσης διάστασης.

Η μέθοδος (PCA) υπολογίζει μία νέα ορθοκανονική βάση για τα δεδομένα, ώστε οι διαστάσεις των δεδομένων σε αυτή να είναι γραμμικά ασυσχέτιστες. Αυτό επιτυγχάνεται με τον υπολογισμό του πρώτου διανύσματος ως αυτό που μπορεί να προσεγγίσει καλύτερα τα δεδομένα υπό την έννοια της ελαχιστοποίησης του μέσου τετραγωνικού λάθους, στην συνέχεια του δεύτερου καλύτερου για τον σκοπό αυτό, το οποίο όμως είναι ορθογώνιο με το πρώτο, του τρίτου το οποίο είναι ορθογώνιο με τα άλλα δύο κ.ο.κ. Έτσι υπολογίζονται τελικά τα διανύσματα τα οποία ονομάζονται principal components και τα οποία είναι γραμμικοί συνδυασμοί των αρχικών μεταβλητών. Το χαρακτηριστικό των διανύσματων αυτών είναι πως η μέγιστη ποσότητα πληροφορίας βρίσκεται στα λίγα πρώτα διανύσματα. Έτσι, μπορούμε επιτυχώς να μειώσουμε την διαστατικότητα χρηστώντας τις δύο ή τρεις πρώτες συνιστώσες με όσο το δυνατόν λιγότερη απώλεια πληροφορίας.

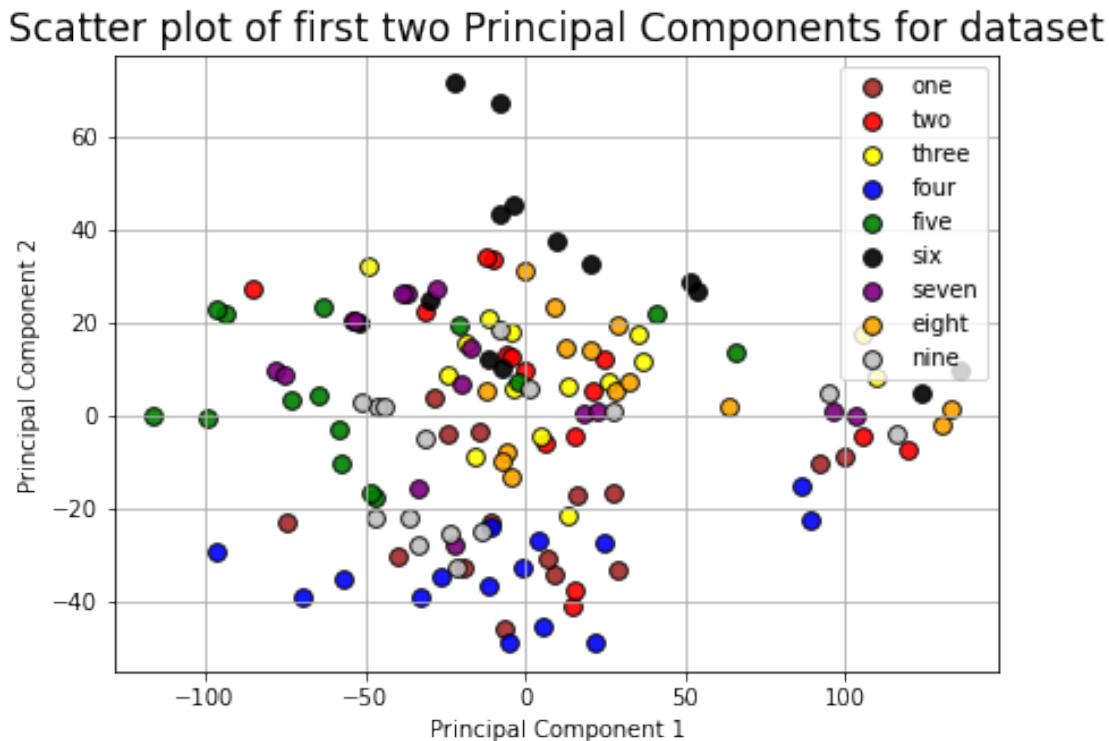
Αρχικά μελετάμε την περίπτωση που λαμβάνουμε τις κύριες 2 συνιστώσες. Λαμβάνουμε τα εξής αποτελέσματα:

First component contains 59.49% of the total information.

Second component contains 11.59% of the total information.

The 2 components combined contain 71.08% of the total information.

Γραφικά έχουμε το εξής αποτέλεσμα:



Στην συνέχεια ακολουθεί η περίπτωση των 3 κύριων συνιστωσών. Τα αποτελέσματα είναι τα εξής:

First component contains 59.49% of the total information.

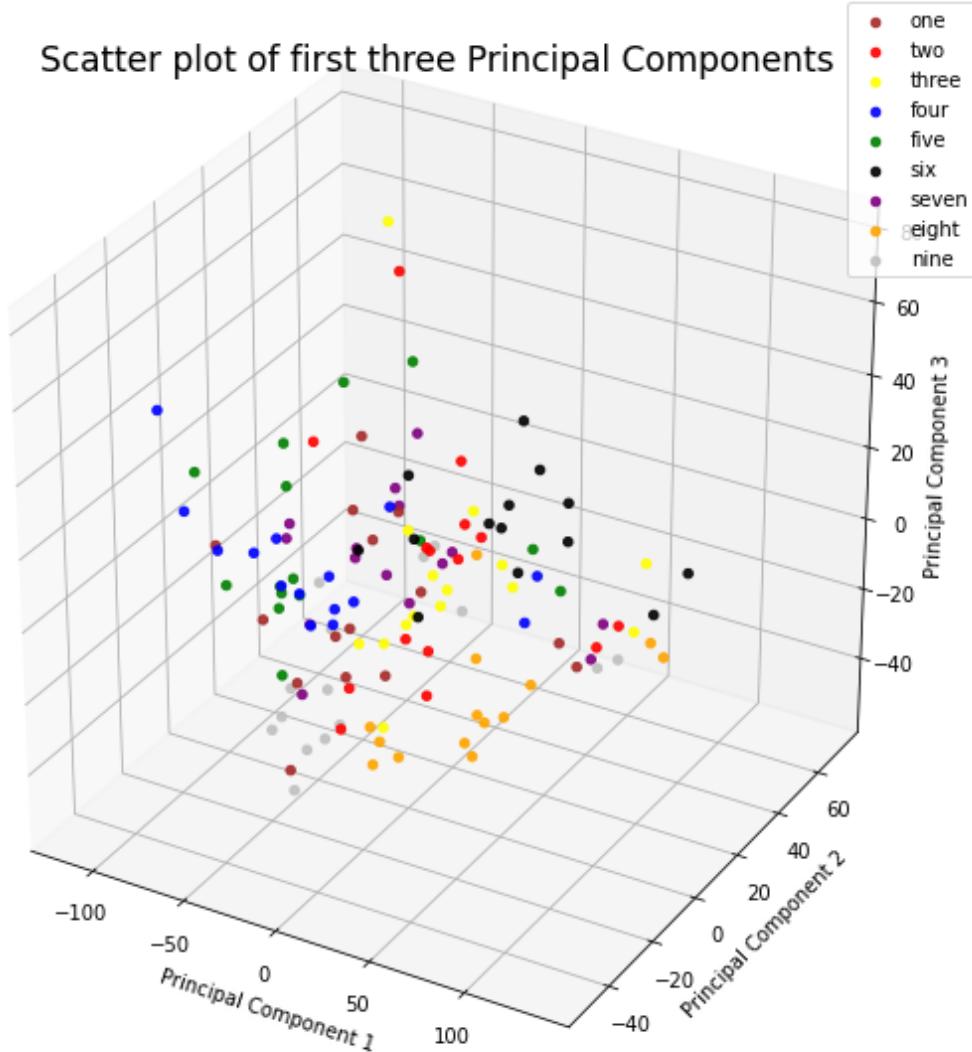
Second component contains 11.59% of the total information.

Third component contains 10.79% of the total information.

The 3 components combined contain 81.87% of the total information.

Γραφικά έχουμε το εξής αποτέλεσμα:

Scatter plot of first three Principal Components



Συγκρίνοντας αρχικά το 2D scatter plot για τα 2 PCA components με αυτό των δύο πρώτων χαρακτηριστικών στο προηγούμενο βήμα μπορούμε να παρατηρήσουμε ότι τώρα, αν και ακόμα υπάρχουν επικαλύψεις στις περιοχές που καταλαμβάνουν τα διάφορα ψηφία, αυτές είναι πιο εύκολα διαχρίσιμες. Αυτό οφείλεται στο γεγονός πως τα δύο πρώτα PCA components συμπεριλαμβάνουν μεγάλο ποσοστό της αρχικής πληροφορίας και είναι πολύ πιο αντιπροσωπευτικά για το dataset μας από την επιλογή δύο τυχαίων από τα χαρακτηριστικά του.

Παρατηρώντας τώρα το τρισδιάστατο scatter plot για τα τρία πρώτα PCA components βλέπουμε πως σε αυτό είναι ακόμα πιο εύκολα διαχωρίσιμη η περιοχή κάθε κλάσης. Επιπλέον, παρατηρούμε πως τα clusters των διαφόρων ψηφίων έχουν αρκετά παρόμοια μορφή με αυτή στο 2D plot, το οποίο είναι λογικό, αφού το τρίτο PCA component περιλαμβάνει ένα 10% επιπλέον της αρχικής διασποράς σε σχέση με το ποσοστό της αρχικής διασποράς που περιλαμβάνουν τα δύο πρώτα components.

Bήμα 7

Στο βήμα αυτό αφού πρώτα χωρίσουμε τα δεδομένα σε train-test με αναλογία 70%-30% και αφού τα κανονικοποιησουμε, χρησιμοποιούμε κάποιους βασεινες αλγορίθμους μηχανικής μάθησης και μελετάμε τα αποτέλεσματα μας. Οι ταξινομητές που χρησιμοποιούμε είναι οι εξής:

- Custom Naive Bayes ταξινομητής της πρώτης εργαστηριακής άσκησης
- Naive Bayes του scikit-learn
- k-nearest neighbors (KNN) του scikit-learn
- Support vector machine (SVM) του scikit-learn
- Multilayer perceptron(MLP) του scikit-learn

Επιλέγουμε, ως μέρος της προεπεξεργασίας των δεδομένων μας μετά την επιλογή χαρακτηριστικών, να τα κανονικοποιήσουμε, προκειμένου να αποφύγουμε προβλήματα στην εκπαίδευση τα οποία μπορεί να προκληθούν από χαρακτηριστικά με πολύ μεγάλες διαφορές στις απόλυτες τιμές τους, οδηγώντας σε ταξινομητές με μη βέλτιστη απόδοση. Για παράδειγμα, ένα χαρακτηριστικό με πολύ μεγάλες τιμές θα έχει μεγαλύτερη επίδραση στον υπολογισμό της απόστασης στον kNN σε σχέση με ένα με μικρές τιμές, χωρίς αυτό να σημαίνει απαραίτητα ότι είναι περισσότερο καθοριστικό για το διαχωρισμό των κλάσεων. Έτσι, χρησιμοποιούμε την κανονικοποίηση ώστε να μετασχηματίσουμε τις τιμές των χαρακτηριστικών και να αμβλύνουμε τις διαφορές αυτές.

Η κανονικοποίηση των χαρακτηριστικών μπορεί να γίνει με 2 βασικούς τρόπους, γνωστούς και από τη στατιστική:

- Με την διαίρεση με τη διαφορά μεγίστου-ελαχίστου (feature scaling) οπότε οι τιμές όλων των χαρακτηριστικών κλιμακώνονται γραμμικά στο διάστημα [0,1]
- Με το z-score (ή standard score) του κάθε χαρακτηριστικού (standardization), που κάνει το χαρακτηριστικό να έχει μέση τιμή μηδέν και διακύμανση μονάδα, σαν την κανονική κατανομή.

Η μετατροπή μεγίστου-ελαχίστου γίνεται με τον τύπο:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Η μετατροπή σε standard score γίνεται με τον τύπο:

$$z = \frac{X - \mu}{\sigma}$$

όπου: μ είναι η μέση τιμή του χαρακτηριστικού και σ η απόκλιση.

Σημειώνουμε ότι η μετατροπή σε standard score είναι απαραίτητη σε πολλούς ταξινομητές για να συμπεριφερθούν σωστά. Επίσης είναι πιο ανθεκτική από την min-max σε τιμές outliers δηλαδή σποραδικές τιμές που είναι πολύ μακριά από τη μέση τιμή και τις υπόλοιπες τιμές του χαρακτηριστικού (η min-max θα συμπιέσει τις περισσότερες τιμές σε ένα μικρό διάστημα).

Από την άλλη, η κλιμάκωση σε [0,1] είναι λιγότερο ευαίσθητη σε πολύ μικρές αποκλίσεις και επίσης σε αραιά (sparse) διανύσματα χαρακτηριστικών (δηλαδή με πολλές μηδενικές τιμές), ενώ η εφαρμογή της διατηρεί τα μηδέν, κάτι που μπορεί να είναι καθοριστικό για την ταχύτητα εκπαίδευσης.

Είναι σημαντικό να αναφέρουμε ότι, προκειμένου να αποφύγουμε την εισαγωγή bias στο σύστημα μας, εφαρμόζουμε στο test set ό,τι μετασχηματισμό κανονικοποίησης κάνουμε στο train set. Τονίζεται ότι δοκιμάσαμε και τα δύο είδη κανονικοποίησης, προκειμένου να δούμε ποια μας οδηγεί σε καλύτερα αποτελέσματα. Επίσης αξίζει να σημειώσουμε ότι πραγματοποιούμε hyperparameter tuning, όπου υπάρχουν υπερ-παράμετροι, με 5-fold cross validation.

Τα αποτελέσματα συνοψίζονται στον παρακάτω πίνακα:

Classifiers	Results					
	Standardization			MinMax scaling		
	Hyper-parameter Tuning	5-Fold Cross Validation Accuracy (%)	Accuracy (%)	Hyper-parameter Tuning	5-Fold Cross Validation Accuracy (%)	Accuracy (%)
Custom Naive Bayes	smoothing = 0	56.20	47.50	smoothing = 0	56.20	47.50
Naive Bayes	smoothing = 0	56.20	47.50	smoothing = 0	56.20	47.50
KNN	algorithm ='auto', n_neighbors =2, weights= 'distance'	66.73	55.00	algorithm ='auto', n_neighbors =2, weights= 'distance'	66.78	57.50
SVM	C = 0.03, gamma= 'scale', kernel= 'linear'	76.55	52.50	C = 0.03, gamma= 'scale', kernel= 'poly'	66.84	50.00
MLP	activation = 'tanh', alpha= 5e-05, learning_rate= 'invscaling', solver= 'lbfgs'	81.87	62.50	activation = 'relu', alpha= 5e-05, learning_rate= 'adaptive', solver= 'lbfgs'	72.05	70.00

Από τα παραπάνω παρατηρούμε ότι καλύτερη επίδοση έχει ο MLP ταξινομητής και στην συνέχεια ακολουθούν ο SVM, KNN και τέλος οι Naive Bayes.

Όσον αφορά την αισθητά χαμηλότερη επίδοση των Naive Bayes ταξινομητών, μπορούμε να εξηγήσουμε αυτή με βάση το γεγονός πως πρόκειται για πιο απλούς ταξινομητές, που κάνουν την υπόθεση γκαουσιανής κατανομής για τα χαρακτηριστικά και μη ύπαρξης συσχέτισης μεταξύ τους. Έτσι, προκύπτουν σφάλματα σε σχέση με την πραγματική κατανομή των χαρακτηριστικών και άρα δεν μπορεί να κωδικοποιηθεί σωστά όλη η πληροφορία για τα ψηφία, ώστε να κατηγοριοποιούνται κατάλληλα σε όλες τις περιπτώσεις.

Συγχρίνοντας τώρα τις επιδόσεις για τις δύο μεθόδους κανονικοποίησης, μπορούμε να παρατηρήσουμε πως η μέθοδος MinMax scaling οδηγεί στις περισσότερες περιπτώσεις σε λίγο καλύτερα αποτελέσματα στο test set.

Τέλος, διαπιστώνουμε πως όλοι οι ταξινομητές που δοκιμάσαμε δεν έχουν πολύ υψηλά ποσοστά ακρίβειας. Αυτό μπορεί εν μέρει να οφείλεται στην μη καταλληλότητα των ταξινομητών αυτών για το συγκεκριμένο πρόβλημα, ή μπορούσε όμως να οφείλεται και σε μία έλλειψη επαρκούς πλήθους δειγμάτων ή την εξαγωγή μη επαρκούς πλήθος χαρακτηριστικών από τα αρχεία ήχου.

Στην συνέχεια, δοκιμάζουμε την προσθήκη επιπλέον ηχιτικών χαρακτηρηστικών στο διάνυσμα κάθε εκφώνησης, προκειμένου να βελτιώσουμε τις επιδόσεις των ταξινομητών. Συγκεκριμένα, προσθέτουμε

τα εξής χαρακτηριστικά:

- Zero-crossing rate:

Το πλήθος μηδενισμών του σήματος της φωνής ανά ένα παράθυρο. Χρησιμοποιούμε μήκος παραθύρου 0.25 ms και βήμα 10 ms.

- Spectral rolloff:

Η συχνότητα κάτω από την οποία βρίσκεται το 85% της ενέργειας του πλάτους του φάσματος του σήματος για κάθε παράθυρο. Χρησιμοποιούμε μήκος παραθύρου 25 ms και βήμα 10 ms.

- Spectral centroid:

Η συχνότητα γύρω από την οποία συγκεντρώνεται η ενέργεια του spectrum, για κάθε παράθυρο. Χρησιμοποιούμε ξανά μήκος παραθύρου 25 ms και βήμα 10 ms.

- Spectral flux:

Το τετράγωνο του αθροίσματος πάνω στην συχνότητα των διαφορών διαδοχικών δειγμάτων του πλάτους του φάσματος.

- LPCs:

Οι linear prediction coefficients που προκύπτουν από την μέθοδο γραμμικής πρόβλεψης των δειγμάτων του σήματος.

- Chroma features:

Το χρωμόγραμμα για το ηχητικό σήμα. Το χαρακτηριστικό αυτό χρησιμοποιείται κυρίως για μουσικά σήματα. Ωστόσο, παρατηρήσαμε πως η χρήση του βελτιώνει την επίδοση πολλών εκτιμητών μας.

Για τα επιμέρους χαρακτηριστικά δοκιμάσαμε την προσθήκη διαφορετικών συνδυσμών από αυτά, ενώ δοκιμάσαμε και άλλα χαρακτηριστικά τα οποία δεν επέφεραν βελτιώσεις στις επιδόσεις των ταξινομητών. Τελικά, καταλήξαμε στην προσθήκη του συνόλου των παραπάνω χαρακτηριστικών (σε συνδυασμό με τα MFCCs, deltas, delta-deltas). Αφού λοιπόν προσθέσαμε τα χαρακτηριστικά αυτά, χωρίσαμε εκ νέου τα δεδομένα μας σε train και test, τα κανονικοποιήσαμε, και προχωρήσαμε στην εκπαίδευση των εκτιμητών που εξετάσαμε και νωρίτερα, με χρήση ξανά 5-fold cross-validation για τον προσδιορισμό των βέλτιστων υπερπαραγμέτρων. Τέλος, προχωρήσαμε σε αξιολόγηση της επίδοσης των μετρητών με χρήση της μετρικής Accuracy. Η αξιολόγηση έγινε με χρήση 5-fold cross Validation στο train set, καθώς με υπολογισμό του accuracy στο test set. Τα αποτελέσματα παρουσιάζονται στον πίνακα που ακολουθεί:

Classifiers	Results					
	Standardization			MinMax scaling		
	Hyper-parameter Tuning	5-Fold Cross Vali-dation Accuracy (%)	Accuracy (%)	Hyper-parameter Tuning	5-Fold Cross Vali-dation Accuracy (%)	Accuracy (%)
Custom Naive Bayes	smoothing = 0	59.42	52.50	smoothing = 0	59.42	52.50
Naive Bayes	smoothing = 0	59.42	52.50	smoothing = 0	59.42	52.50
KNN	algorithm ='auto', n_neighbors =2, weights='distance'	68.83	62.50	algorithm ='auto', n_neighbors =2, weights='distance'	67.72	65.00
SVM	C = 0.03, gamma= 'scale', kernel= 'linear'	77.66	67.50	C = 0.05, gamma= 'scale', kernel= 'poly'	73.27	65.00
MLP	activation = 'iden-tity', alpha= 0.01, learn-ing_rate= 'invscaling', solver= 'lbfgs'	76.55	70.00	activation = 'iden-tity', alpha= 0.01, learn-ing_rate= 'constant', solver= 'lbfgs'	81.75	80.00

Παρατηρούμε πως η προσθήκη των επιπλέον χαρακτηρηστικών βελτίωσε σημαντικά τις επιδόσεις όλων των ταξινομητών. Όσον αφορά δε την σύγκριση των επιδόσεων των διαφόρων ταξινομητών μεταξύ τους, παρατηρούμε ότι αυτή δεν άλλαξε σε σχέση με πριν, δηλαδή εξακολουθεί ο MLP να είναι ο βέλτιστος για το παρόν πρόβλημα, με τους SVM, kNN και Naive Bayes να ακολουθούν, σε αυτή την σειρά. Σημειώνουμε τέλος ότι, αν και οι επιδόσεις των ταξινομητών μας έχουν βελτιωθεί σημαντικά, το accuracy εξακολουθεί να μην είναι επαρκώς μεγάλο, γεγονός που θα μας οδηγούσε ενδεχομένως στην δοκιμή διαφορετικών μοντέλων ή στην αναζήτηση μεγαλύτερου dataset.

Βήμα 8

Στο βήμα καλούμαστε να υλοποιήσουμε ένα RNN το οποίο με είσοδο 10 σημεία ενός ημιτόνου (συχνότητας 40Hz) πρέπει να προβλέπει τα αντίστοιχα 10 σημεία ενός συνημιτόνου (συχνότητας 40Hz). Για τον σκοπό αυτό δημιουργούμε αρχικά 1000 διαφορετικά δείγματα, το καθένα από τα οποία αποτελείται από 10 σημεία ενός ημιτόνου συχνότητας f=40Hz και τυχαίου πλάτους μεταξύ 0 και 1. Για καθένα από τα δείγματα αυτά δημιουργούμε και το αντίστοιχο συνημίτονο, το οποίο και επιθυμούμε να προβλέπει το RNN που θα υλοποιήσουμε.

Αφού λοιπόν έχουμε δημιουργήσει το αρχικό μας dataset, χωρίζουμε αυτό σε train και test, και δημιουργούμε pythorch Dataloaders για την διαχείρισή του.

Έχοντας λοιπόν έτοιμα τα train και test δεδομένα και labels (συνημίτονα), προχωράμε στην υλοποίηση του RNN.

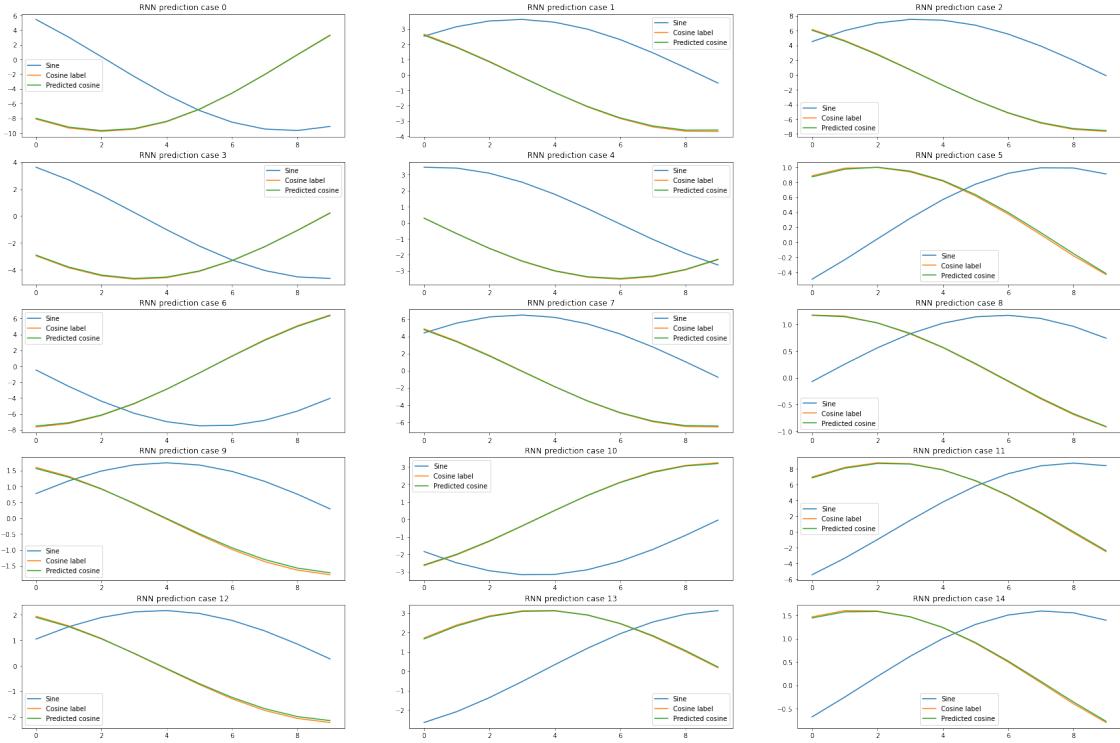
Τα RNN αποτελούν έναν τύπο νευρωνικού που χρησιμοποιείται συχνά σε εφαρμογές αναγνώρισης φωνής, μετάφρασης και επεξεργασίας φυσικής γλώσσας. Χαρακτηριστικό του τύπου αυτού νευρωνικού είναι πως χρησιμοποιούν πληροφορία από προηγούμενα inputs και outputs για τον προσδιορισμό του επόμενου input και output.

Στην παρούσα εφαρμογή υλοποιούμε ένα RNN το οποίο διαθέτει ένα hidden layer διάστασης 100, ενώ έχει διάσταση εξόδου 10, όσα δηλαδή και τα δείγματα κάθε ακολουθίας που πρέπει να προβλέψει. Χρησιμοποιούμε κατά την εκπαίδευση batch size ίσο με 500 και learning rate 0.01. Τέλος, εκπαιδεύουμε το νευρωνικό σε 1875 epochs και παρατηρούμε τα loss πάνω στο train set και πάνω στο test set κατά την εκπαίδευση. Σημειώνουμε ότι χρησιμοποιούμε το MSE ως loss function και επιλέγουμε optimizer Adam.

Ενδεικτικά παρουσιάζουμε ένα στιγμότυπο των losses που τυπώνουμε κάθε 100 training epochs:

```
Train loss 16.113449096679688 Test loss: 15.19412899017334
Train loss 15.12734603881836 Test loss: 13.63972282409668
Train loss 0.0012344097485765815 Test loss: 0.001329085323959589
Train loss 0.0013083505909889936 Test loss: 0.0013277233811095357
Train loss 0.0005148839554749429 Test loss: 0.0005841858219355345
Train loss 0.000448511797003448 Test loss: 0.0005662029143422842
Train loss 0.025352444499731064 Test loss: 0.007661106064915657
Train loss 0.006927579175680876 Test loss: 0.001995959784835577
:
Train loss 0.00013148011930752546 Test loss: 0.00020224963373038918
Train loss 0.00012083475303370506 Test loss: 0.00016492072609253228
Train loss 0.00010376450518378988 Test loss: 0.00017568044131621718
```

Αφού ολοκληρωθεί η εκπαίδευση του νευρωνικού εξετάζουμε την επίδοσή του στην πρόβλεψη των ακολουθιών συνημίτονου από τις ακολουθίες ημιτόνου. Παρουσιάζουμε ενδεικτικά 15 περιπτώσεις:

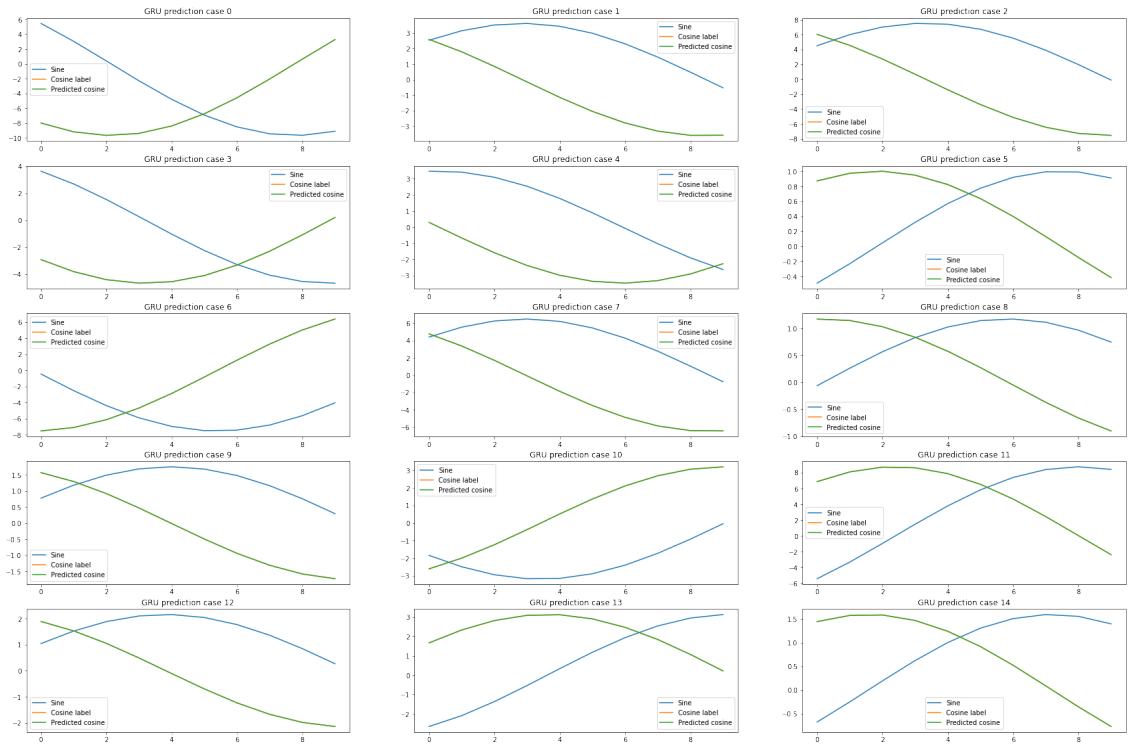


Παρατηρούμε πως το RNN που δημιουργήσαμε προβλέπει τις ακολουθίες συνημιτόνου με πολύ μεγάλη ακρίβεια, αφού οι προβλεπόμενες ακολουθίες συμπίπτουν σχεδόν με τις επιθυμητές.

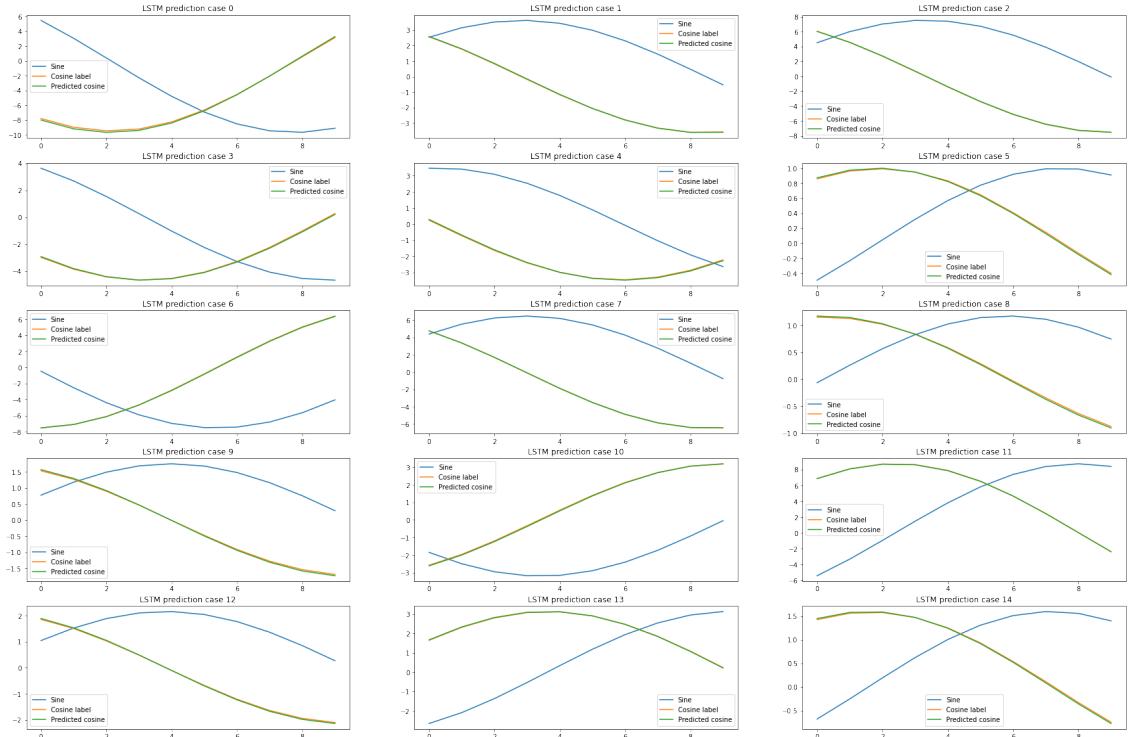
Δοκιμάζουμε στην συνέχεια να χρησιμοποιήσουμε τις μονάδες GRU και LSTM αντί του απλού RNN. Τα GRU και LSTM χρησιμοποιούνται συχνά αντί για το απλό RNN, καθώς είναι ειδικά σχεδιασμένα για προβλήματα που απαιτούν την μάθηση long-term χρονικών ακολουθιών. Συγκεκριμένα, καθώς το gradient του loss function μειώνεται εκθετικά με τον χρόνο, είναι δύσκολο τα απλά RNN να διατηρήσουν πληροφορία που προηγήθηκε αρκετά πριν στον χρόνο. Το πρόβλημα αυτό λύνεται από τα GRU και LSTM, τα οποία διαθέτουν ειδικές μονάδες gates οι οποίες ελέγχουν ποιες πληροφορίες θα ψυμάται το νευρωνικό και πότε θα τις ξεχνά.

Η υλοποίηση των GRU και LSTM είναι αρκετά απλή, καθώς αρκεί η αντικατάσταση του δομικού στοιχείου της κλάσης του RNN μοντέλου από RNN σε GRU ή LSTM (οι υπόλοιπες αλλαγές είναι πολύ μικρές και φαίνονται στον κώδικα μας). Έτσι, προκύπτουν τελικά τα εξής αποτελέσματα:

- Για το GRU:



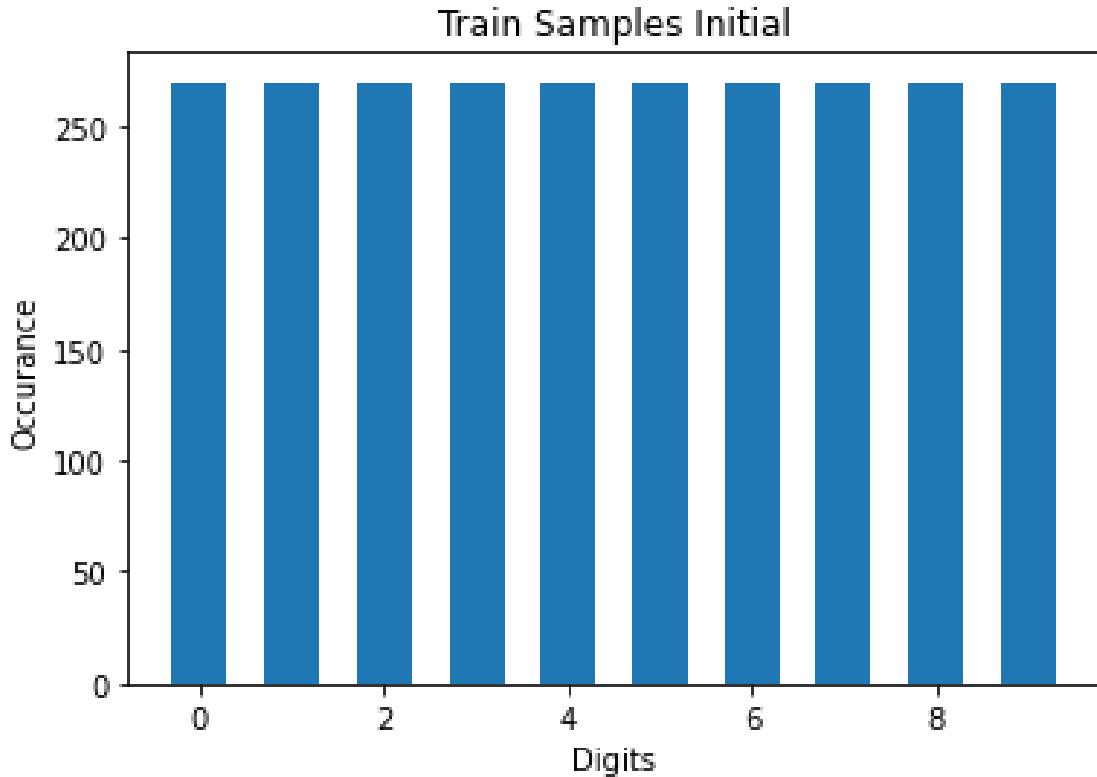
- Για το LSTM:



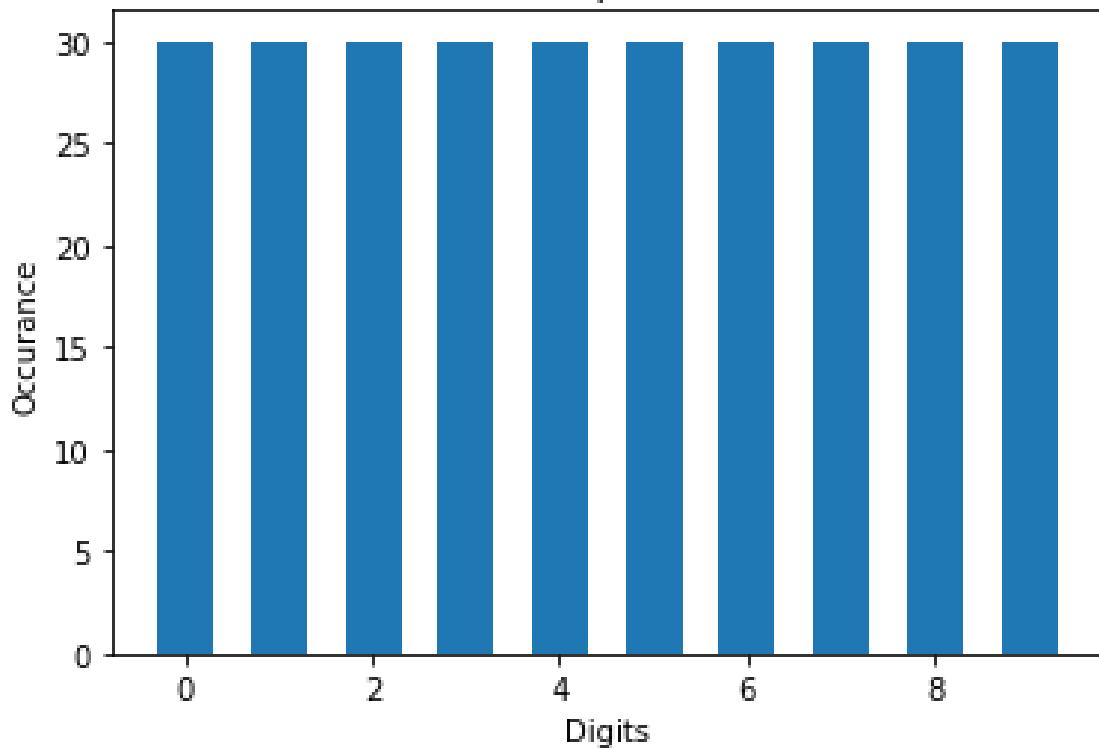
Παρατηρούμε πως έχουμε πολύ καλά αποτελέσματα για τα GRU και LSTM. Βέβαια, τα αποτελέσματα ήταν ήδη πολύ καλά και με το απλό RNN για το συγκεκριμένο πρόβλημα, και επομένως αυτό δεν απαιτεί την χρήση GRU ή LSTM. Ωστόσο, σε πιο σύνθετα προβήματα και μεγαλύτερες χρονικά ακολουθίες θα περιμένανε τα GRU και LSTM να έχουν καλύτερες επιδόσεις.

Βήμα 9

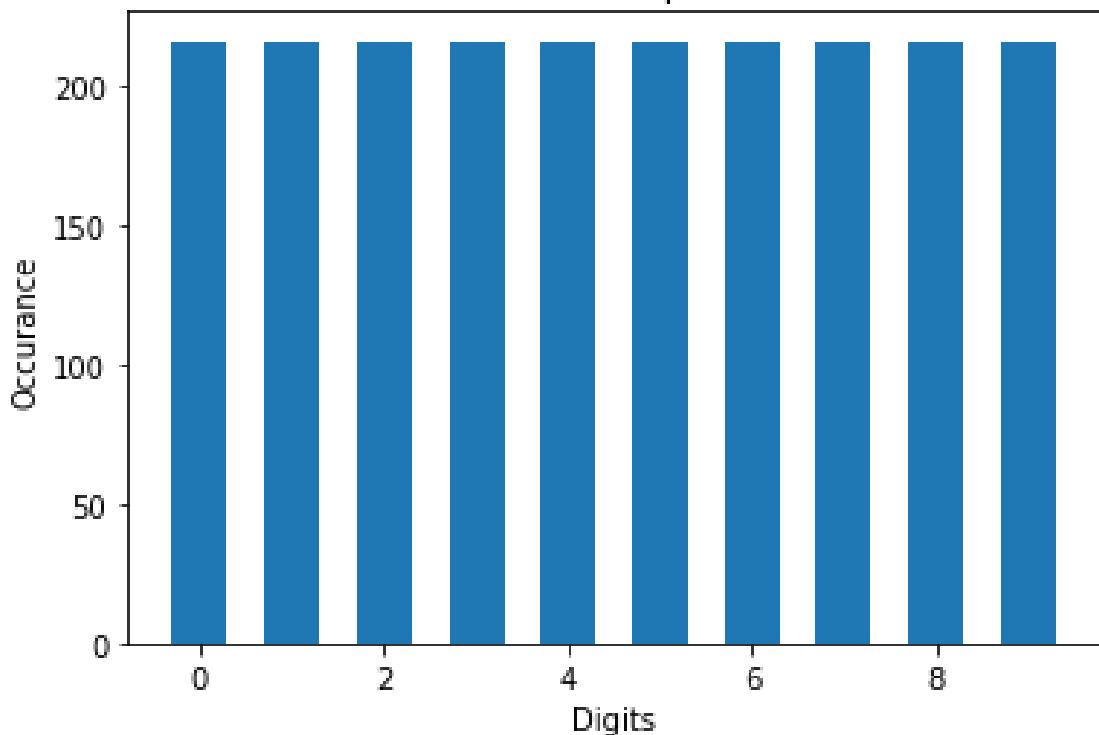
Από το βήμα αυτό και στα επόμενα βήματα ασχολούμαστε με ένα μεγαλύτερο dataset, το Free Spoken Digit Dataset. Το dataset αυτό αποτελείται από 3000 δείγματα και καθένα από αυτά έχει συχνότητα 8 kHz. Από τις ηχογραφήσεις εξάγουμε 6 MFCC χαρακτηριστικά για κάθε χρονικό παράθυρο του σήματος φωνής. Έπειτα χωρίζουμε τα train δεδομένα σε training και validation set με ποσοστό 80%-20%. Επισημαίνεται ότι προκειμένου να διατηρηθεί ίδιος ο αριθμός των διαφορετικών ψηφίων σε κάθε set θέτουμε κατά τον χωρισμό την παράμετρο stratify = y_train. Σημειώνουμε επίσης ότι χρησιμοποιούμε τον κώδικα που βρίσκεται στο βοηθητικό αρχείο parser.py. Ετσι παίρνουμε τα παρακάτω διάγραμμα για το πλήθος των ψηφίων που έχουμε σε κάθε ένα από τα 3 set που δημιουργήσαμε και τις αρχικές κατανομές:



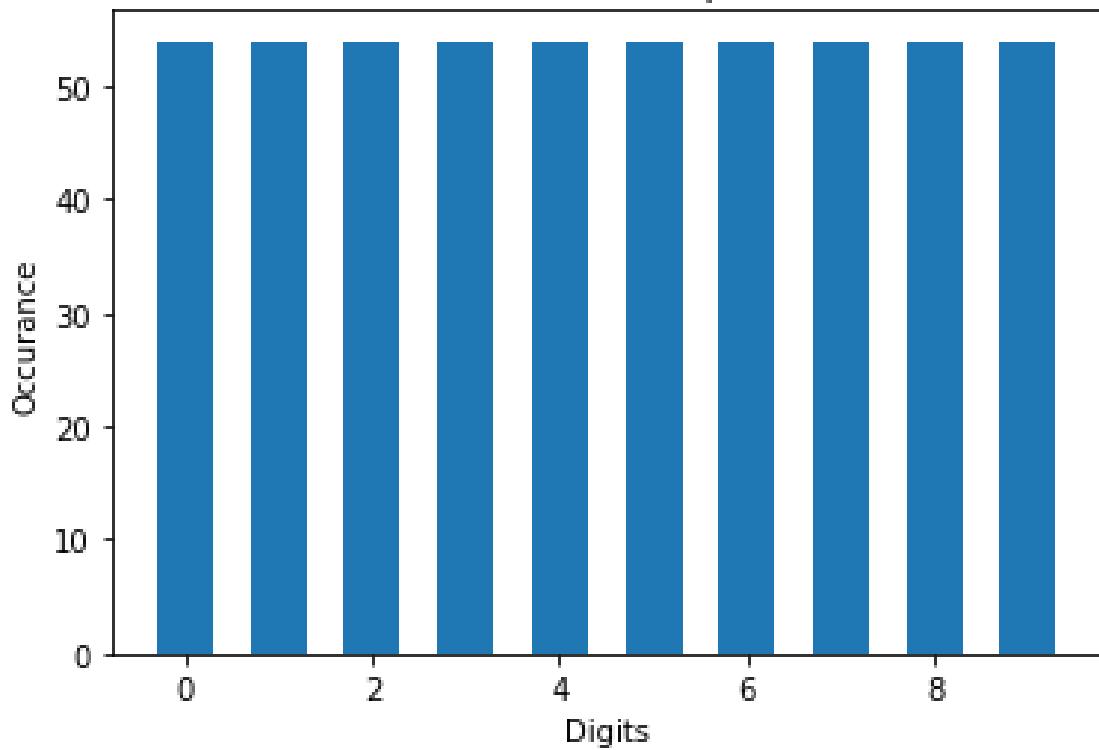
Test Samples Initial



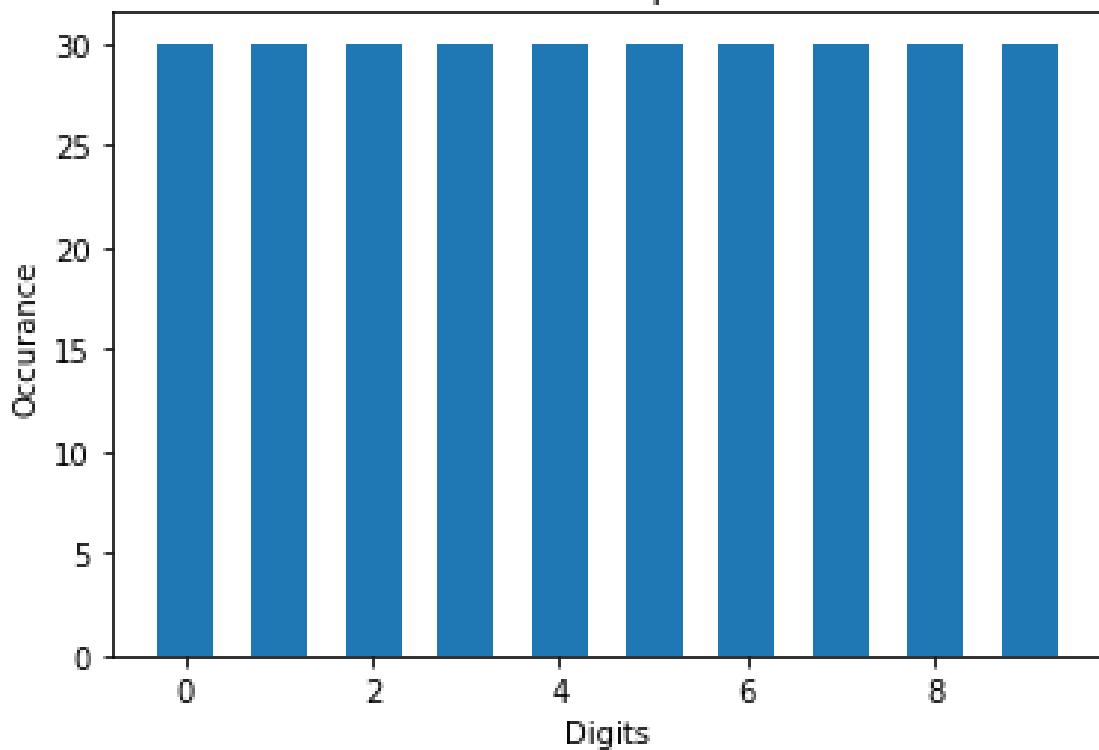
Train Samples



Validation Samples



Test Samples



Βήμα 10

Στο ερώτημα αυτό θα πραγματοποιήσουμε αναγνώριση των ψηφίων με τη χρήση GMM-HMM με τη βιόγλια της βιβλιοθήκης pomegranate. Αρχικά αρχικοποιήσαμε ένα GMM-HMM μοντέλο για κάθε ψηφίο. Το GMM χρησιμοποιείται για την μοντελοποίηση της κατανομής των διανυσμάτων χαρακτηριστικών με πολυδιάστατες γκαουσιανές. Μέσω του αλγορίθμου Expectation Maximization υπολογίζουμε το διάνυσμα μέσων τιμών και τον πίνακα συνδιακύμανσης για κάθε μία από τις γκαουσιανές. Το HMM είναι μία μαρκοβιανή αλυσίδα με κρυφές καταστάσεις στην οποία οι μεταβάσεις μεταξύ καταστάσεων μοντελοποιούν την αλλαγή φωνήματος. Το κάθε μοντέλο είναι left-right. Αυτό σημαίνει ότι χρησιμοποιούμε ένα άνω τριγωνικό και στοχαστικό πίνακας μετάβασης. Αυτός με βάση τα δεδομένα της εκφώνησης θα έχει την εξής μορφή:

$$trans_mat = \begin{bmatrix} 0.5 & 0.5 & 0 & \dots & 0 \\ 0 & 0.5 & 0.5 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Επιπλέον, εκτός από τον πίνακα μετάβασης κάθε μοντέλο GMM-HMM έχει τις εξής μεταβλητές:

- X: διάνυσμα χαρακτηριστικών από ένα ψηφίο
- n_stages: ο αριθμός των HMM καταστάσεων
- n_mixtures: ο αριθμός των Gaussians. Όταν το n_mixtures είναι μεγαλύτερο του 1 χρησιμοποιούμε GMM αλλιώς 1d Gaussian.
- dists: λίστα των κατανομών πιθανότητας για τις καταστάσεις των HMM.
- gmm: κατά πόσο θα χρησιμοποιήσουμε GMM ή plain Gaussian
- starts: αρχικός πίνακας πιθανοτήτων με

$$p_i = 0g_{i1} \neq 1k_{i1}1g_{i1} = 1$$

- ends: τελικές πιθανότητες
- models = μοντέλο για το κάθε ψηφίο που θα χρησιμοποιηθεί.

Δημιουργούμε μία κλάση για το μοντέλο HMM-GMM στον κώδικά μας, στην οποία ορίζονται οι παραπάνω παράμετροι και αρχικοποιείται το μοντέλο. Η κλάση περιλαμβάνει επιπλέον τις μεθόδους fit, η οποία εκπαιδεύει το μοντέλο για δοσμένα δεδομένα, και predict, η οποία επιστρέφει επιστρέφει τον λογάριθμο πιθανότητας για που προκύπτει με τον αλγόριθμο Viterbi, με είσοδο ένα δείγμα.

Στην συνέχεια, δημιουργούμε μία συνάρτηση create_hmms στον κώδικά μας με την οποία αρχικοποιούμε 10 τέτοια μοντέλα GMM-HMM (ένα για κάθε ψηφίο) και τα προσθέτουμε σε μία λίστα που ονομάζουμε hmms. Σημειώνουμε ότι η συνάρτηση αυτή δέχεται ως παράμετρο το πλήθος από gaussian mixtures που θα χρησιμοποιήσουμε καθώς και το πλήθος καταστάσεων που θα έχει το κάθε μοντέλο. Γενικά, για κάθε εκφώνηση, οι τιμές αυτές θα μπορούσαν να είναι διαφορετικές για να έχουμε βέλτιστη μοντελοποίηση. Χάριν όμως απλότητας, και καυνώς το πρόβλημα αναγνώρισης μεμονομένων ψηφίων είναι αρκετά απλό, κάτι τέτοιο δεν επιλέγεται, και αρκούμαστε στην χρήση ίδιου πλήθους gaussian mixtures και καταστάσεων για όλα τα μοντέλα.

Βήμα 11

Στο βήμα αυτό εκπαιδεύουμε τα 10 μοντέλα με χρήση του αλγορίθμου Expectation Maximization. Για τον σκοπό αυτό δημιουργούμε την συνάρτηση fit_hmms, η οποία για κάθε μοντέλο καλεί την μέθοδο fit της κλάσης του. Για την εκπαίδευση του κάθε μοντέλου χρησιμοποιούμε όλα τα διαθέσιμα δεδομένα για το ψηφίο αυτό.

Βήμα 12

Στο βήμα αυτό αναζητούμε τις παραμέτρους εκείνες (πλήθος καταστάσεων και gaussian mixtures) οι οποίες μας οδηγούν στα βέλτιστα αποτελέσματα αναγνώρισης ψηφίων πάνω στο validation set. Δημιουργούμε αρχικά μία συνάρτηση evaluate_hmms η οποία κάθε δείγμα υπολογίζει το log probability και των 10 μοντέλων καλώντας την μέθοδο predict αυτών, και επιλέγει ως πρόβλεψη το ψηφίο στο οποίο αντιστοιχεί το μοντέλο με την μεγαλύτερη log probability. Αφού επαναλάβει το ίδιο για όλα τα δείγματα υπολογίζει το accuracy το οποίο και επιστρέφει.

Έχοντας λοιπόν δημιουργήσει μία συνάρτηση που αξιολογεί τα μοντέλα μας, πρέπει να βρούμε τις παραμέτρους εκείνες, οι οποίες καθιστούν το μοντέλο βέλτιστο. Για τον σκοπό αυτό δημιουργούμε μια συνάρτηση grid_search η οποία δοκιμάζει όλους τους συνδυασμούς διαφορετικών τιμών για το πλήθος γκαουσισιανών, το πλήθος καταστάσεων, καθώς και το μέγιστο πλήθος επαναλήψεων κατά την εκπαίδευση των μοντέλων, που λαμβάνει ως παραμέτρους σε μορφή λιστών. Εξετάζουμε αριθμό καταστάσεων από 1 έως 4 και αριθμό γκαουσισιανών από 1 έως 5. Για τα διαφορετικά μοντέλα που προκύπτουν με n_stages = [1,2,3,4], n_mixtures = [1,2,3,4,5], n_iter = [5,10,50,100] κάνουμε fit και μέσω της evaluate_hmms εκτελείται ο αλγόριθμος Viterbi για τον υπολογισμό της πιο συχνής ακολουθίας καταστάσεων. Σημειωνούμε ότι δεν δοκιμάζουμε πολύ μεγάλα πλήθη καταστάσεων και γκαουσισιανών, καθώς δεν είναι αρκετές οι ακολουθίες που διαθέτουμε για την εκπαίδευση μεγαλύτερων μοντέλων. Μερικά ενδεικτικά αποτελέσματα που προκύπτουν μετα την εκπαίδευση για το validation set είναι:

```
HMM with: number of HMM states: 1, number of GMMs: 1, maximum number of iterations: 5 has accuracy: 0.7648148148148148
HMM with: number of HMM states: 1, number of GMMs: 1, maximum number of iterations: 10 has accuracy: 0.7648148148148148
HMM with: number of HMM states: 1, number of GMMs: 1, maximum number of iterations: 50 has accuracy: 0.7648148148148148
HMM with: number of HMM states: 1, number of GMMs: 1, maximum number of iterations: 100 has accuracy: 0.7648148148148148
HMM with: number of HMM states: 1, number of GMMs: 2, maximum number of iterations: 5 has accuracy: 0.8444444444444444
HMM with: number of HMM states: 1, number of GMMs: 2, maximum number of iterations: 10 has accuracy: 0.85
HMM with: number of HMM states: 1, number of GMMs: 2, maximum number of iterations: 50 has accuracy: 0.8388888888888889
HMM with: number of HMM states: 1, number of GMMs: 2, maximum number of iterations: 100 has accuracy: 0.8629629629629629
HMM with: number of HMM states: 1, number of GMMs: 3, maximum number of iterations: 5 has accuracy: 0.85
HMM with: number of HMM states: 1, number of GMMs: 3, maximum number of iterations: 10 has accuracy: 0.8907407407407407
HMM with: number of HMM states: 1, number of GMMs: 3, maximum number of iterations: 50 has accuracy: 0.8722222222222222
HMM with: number of HMM states: 1, number of GMMs: 3, maximum number of iterations: 100 has accuracy: 0.8703703703703703
HMM with: number of HMM states: 1, number of GMMs: 4, maximum number of iterations: 5 has accuracy: 0.8981481481481481
HMM with: number of HMM states: 1, number of GMMs: 4, maximum number of iterations: 10 has accuracy: 0.9185185185185185
HMM with: number of HMM states: 1, number of GMMs: 4, maximum number of iterations: 50 has accuracy: 0.8833333333333333
HMM with: number of HMM states: 1, number of GMMs: 4, maximum number of iterations: 100 has accuracy: 0.9018518518518519
HMM with: number of HMM states: 1, number of GMMs: 5, maximum number of iterations: 5 has accuracy: 0.9407407407407408
HMM with: number of HMM states: 1, number of GMMs: 5, maximum number of iterations: 10 has accuracy: 0.95
HMM with: number of HMM states: 1, number of GMMs: 5, maximum number of iterations: 50 has accuracy: 0.9259259259259259
HMM with: number of HMM states: 1, number of GMMs: 5, maximum number of iterations: 100 has accuracy: 0.9222222222222223
HMM with: number of HMM states: 2, number of GMMs: 1, maximum number of iterations: 5 has accuracy: 0.8740740740740741
HMM with: number of HMM states: 2, number of GMMs: 1, maximum number of iterations: 10 has accuracy: 0.8777777777777778
HMM with: number of HMM states: 2, number of GMMs: 1, maximum number of iterations: 50 has accuracy: 0.8611111111111112
```

Τελικά από το grid search προκύπτει:

Best parameters: number of HMM states: 4, number of GMMs: 5, maximum number of iterations: 5

Για αυτές τις τιμές το accuracy είναι:

Model has accuracy in test set: 0.98

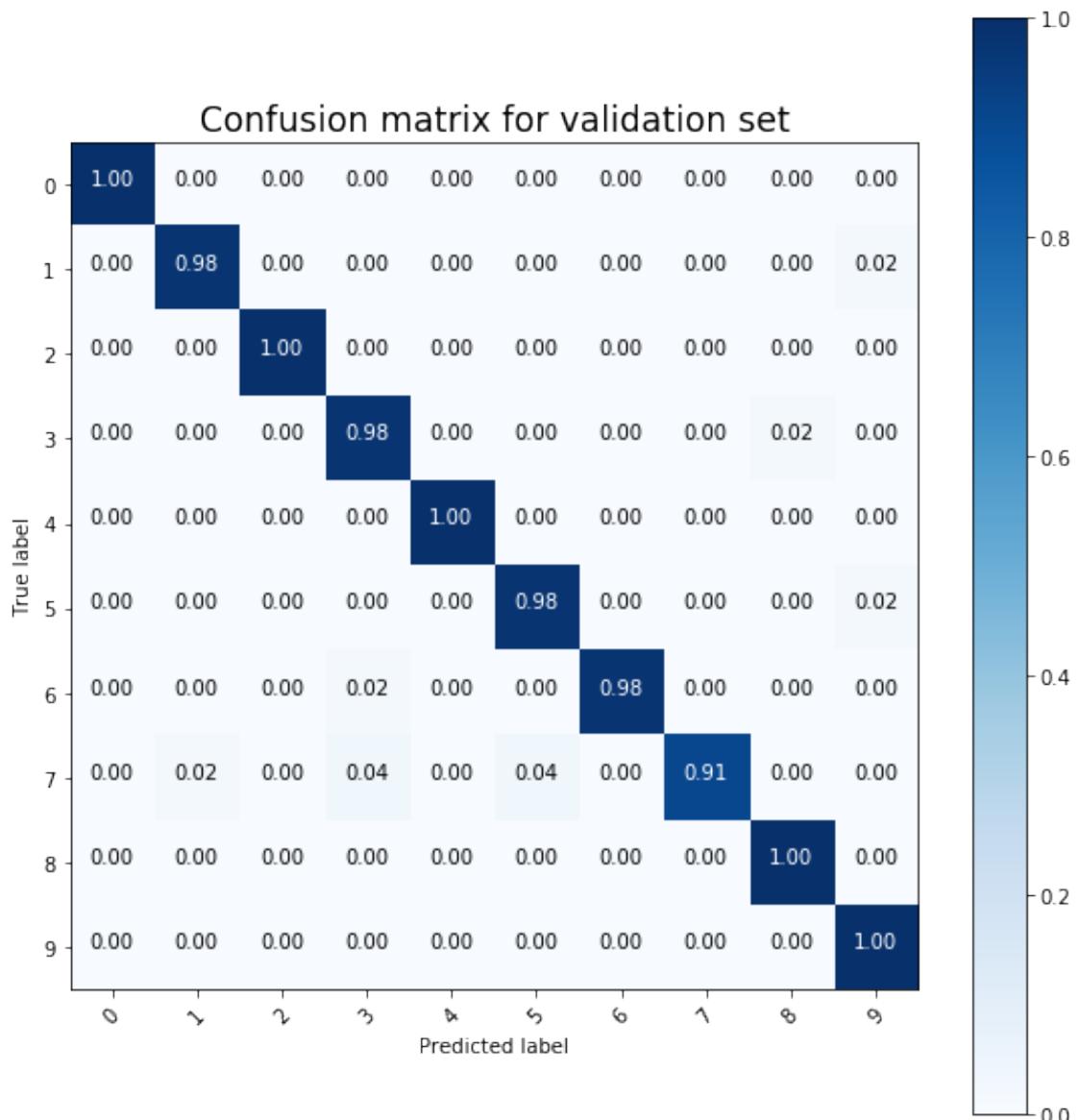
Η διαδικασία του hyperparameter tuning μας βοηθάει να επιλέξουμε τις καλύτερες παραμέτρους για το κάθε παράδειγμα που έχουμε και έτσι να βελτιστοποιήσουμε την επίδοση του μοντέλου μας. Από τα αποτελέσματα μας, όπως αναφέραμε, διαπιστώνουμε ότι χρειαζόμαστε 5 GMMs για βέλτιστη επίδοση. Γενικά, πολλαπλές γκαουσισιανές μας επιτρέπουν να ταιριάζουμε το μοντέλο μας στα δεδομένα μας ακόμα και όταν τα δεδομένα κάθε κλάσης ανήκουν σε διαφορετικά clusters, ενώ μας επιτρέπουν να δουλεύουμε σε δεδομένα στα οποία οι διαφορετικές κλάσεις εμφανίζουν overlap. Όσον αφορά το ακριβές πλήθος αυτών που χρειαζόμαστε, μπορούμε να σχολιάσουμε ότι ενδεχομένως αυτό να είναι

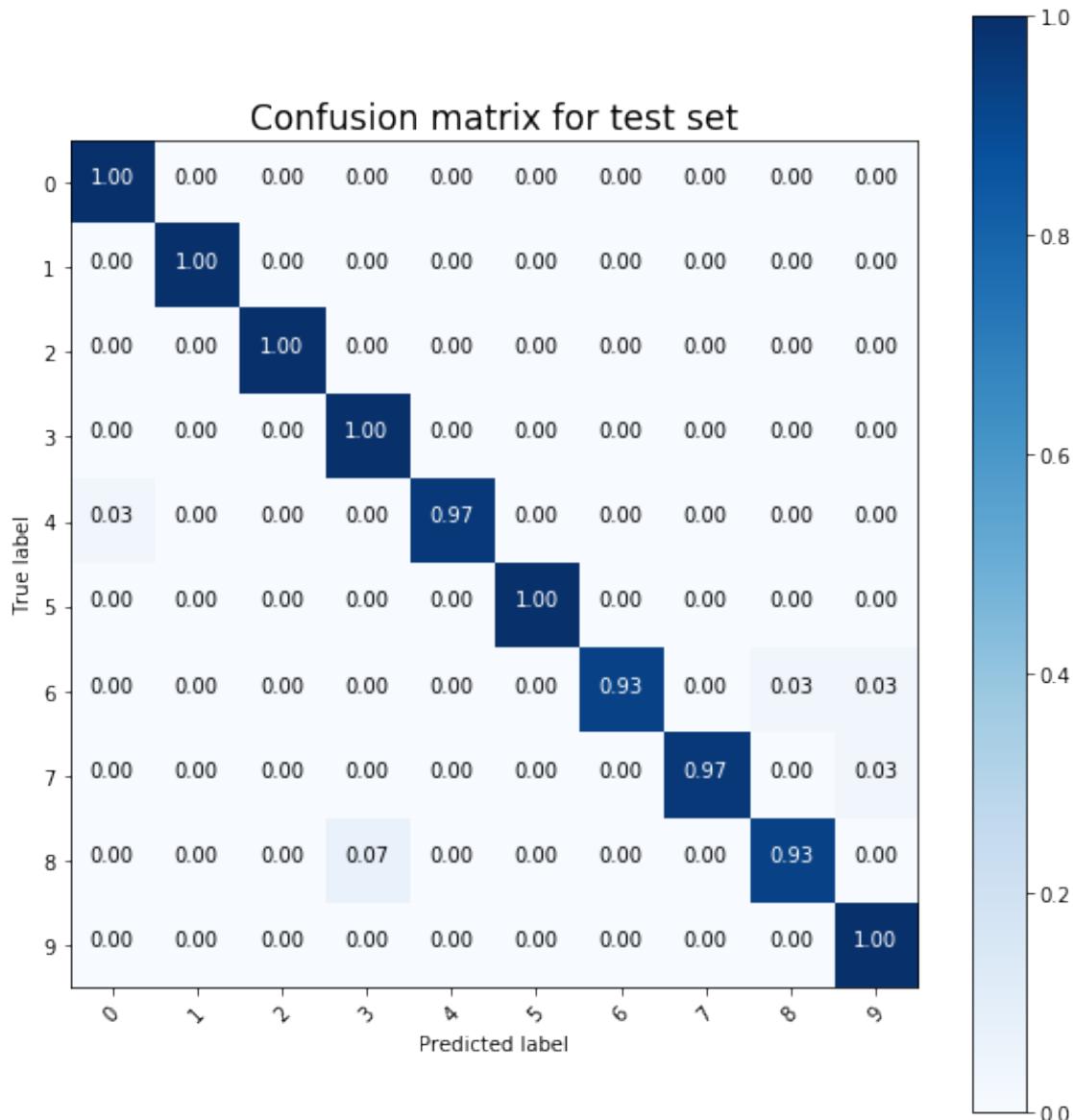
διαφορετικό για κάθε ψηφίο, ωστόσο, με βάση την απλοποίηση του να θεωρήσουμε το ίδιο πλήθος για όλα τα ψηφία την οποία αναφέραμε και νωρίτερα πως πραγματοποίησαμε, προκύπτει τελικά οτι η γενικά βέλτιση περίπτωση είναι για χρήση 5 γκαουσσιανών. Προκύπτει επιπλέον ότι ο μέγιστος αριθμός καταστάσεων που δοκιμάσαμε (4 καταστάσεις) είναι και ο βέλτιστος, καθώς για μεγαλύτερο αριθμό καταστάσεων μπορεί να συμπεριληφθεί περισσότερη πληροφορία. Ωστόσο σημειώνουμε ότι παιρεταίρω αύξηση δεν είναι βέβαιο ότι θα οδηγήσει σε καλύτερα αποτελέσματα αφού τα features των σημάτων φωνής μπορεί να έχουν και άνω όριο στον αριθμό των χαρακτηριστικών που συσχετίζονται.

Αξίζει επιπλέον να σχολιάσουμε ότι χρησιμοποιούμε κατά την εύρεση των βέλτιστων υπερπαραμέτρων μόνο το validation κα κά όχι το test set. Ο λόγος για αυτό είναι ο εξής: Κατά την διαδικασία βέλτιστοποίησης υπάρχει ο κίνδυνος να οδηγηθούμε σε overfitting, υπολογίζοντας τις βέλτιστες υπερπαραμέτρους που οδηγούν σε πολύ καλά αποτελέσματα για τα συγκεκριμένα δεδομένα, και χρησιμοποιώντας μεγάλο πλήθος επαναλήψεων κατά την εκπαίδευση του μοντέλου (το μέγιστο πλήθος επαναλήψεων αποτελεί μία από τις υπερπαραμέτρους που εξετάζουμε). Προκειμένου να μπορούμε λοιπόν να εξετάσουμε την ικανότητα του μοντέλου μας να γενικεύει σε δεδομένα που δεν έχει 'ξαναδεί' και να βεβαιωθούμε πως είναι όσο το δυνατό λιγότερο biased, διατηρούμε το test set για την τελική μόνο αξιολόγηση του μοντέλου, το οποίο έχουμε ήδη βελτιστοποιήσει με χρήση του validation set.

Βήμα 13

Στο βήμα αυτό προκειμένου να αξιολογήσουμε καλύτερα τα αποτελέσματα του μοντέλου μας σχεδιάζουμε τον Confusion Matrix για το validation και το test set. Κάθε confusion matrix έχει μέγεθος (10*10) (όσα και τα ψηφία μας) και το στοιχείο του (i,j) ισούται με τις φορές που το μοντέλο προέβλεψε j ενώ η σωστή απάντηση ήταν i. Τα αποτελέσματα είναι τα εξής:





Παρατηρούμε ότι στο validation set προκύπτουν λάθη στις κλάσεις 1,3,5,6,7, ενώ στο test set στις κλάσεις 4,6,7,8, ενώ διαφοροποίηση υπάρχει και στο ποια κλάση δίνεται ως απόντηση λανθασμένα, όταν προκύπτει λάθως. Επομένως, μπορούμε να πούμε πως τα λάθη είναι σχετικά "τυχαία", δηλαδή το μοντέλο μας δεν εμφανίζει ιδιαίτερη τάση να κάνει ένα συγκεκριμένο λάθος και δεν είναι biased. Ακόμα και στις κλάσεις πάντως που παρουσιάζονται λάθη, αυτά είναι αρκετά σπάνια, το οποίο είναι αναμενόμενο, δεδομένου ότι έχουμε accuracy 0.98.

Βήμα 14

Στο ερώτημα αυτό πραγματοποιούμε αναγνώριση των φηφίων με χρήση LSTM μοντέλων.

- Αρχικά, υλοποιούμε το FrameLevelDataset προσθέτοντας κώδικα στον δισμένο κώδικα του εργαστηρίου, και φορτώνουμε τα train, validation, test δεδομένα σε FrameLevelDataset. Υστερα δημιουργούμε Dataloaders για τα train, validation, test δεδομένα μας, στους οποίους θέσαμε Batch size ίσο με 16. Η τιμή αυτή προέχυψε πειραματικά μετά από διάφορες δοκιμές. Στην συνέχεια, χρησιμοποιώντας το βοηθητικό κώδικα που βρίσκεται στο lstm.py υλοποιούμε

ένα αναδρομικό νευρωνικό δίκτυο, στο οποίο προσθέτουμε ήδη τις επιλογές για ύπαρξη dropout, καθώς και για bidirectional LSTM, οι οποίες είναι χρήσιμες σε επόμενα ερωτήματα.

2. Στο ερώτημα αυτό αρχικοποιούμε ένα LSTM μοντέλο. Χρησιμοποιούμε:

- αριθμό hidden επιπέδων = 2
- input dimension = 6 (αριθμός των MFCCs)
- hidden layer dimension = 128
- output dimension = 10 (όσες και οι κλάσεις των ψηφίων)
- learning rate = 0.001
- αριθμό εποχών = 30
- loss_fuction = nn.CrossEntropyLoss()
- optimizer = torch.optim.Adam()

Σημειώνουμε ότι οι παραπάνω τιμές προέκυψαν πειραματικά, μετά από ένα πλήθος δοκιμών.

3. Στην συνέχεια, εκπαιδεύουμε το δίκτυο και τυπώνουμε το mean training loss σε κάθε εποχή. Ενδεικτικά έχουμε:

Epoch 0: Mean training loss per epoch: 1.534659426300614

Epoch 1: Mean training loss per epoch: 0.6881914898201271

Epoch 2: Mean training loss per epoch: 0.4721988879420139

Epoch 3: Mean training loss per epoch: 0.33459952628171

Epoch 4: Mean training loss per epoch: 0.19093620016894958

Epoch 5: Mean training loss per epoch: 0.184859105406536

Epoch 6: Mean training loss per epoch: 0.12498832461596639

Epoch 7: Mean training loss per epoch: 0.08783758872499069

Epoch 8: Mean training loss per epoch: 0.13687668833536681

Epoch 9: Mean training loss per epoch: 0.08229190827933726

⋮

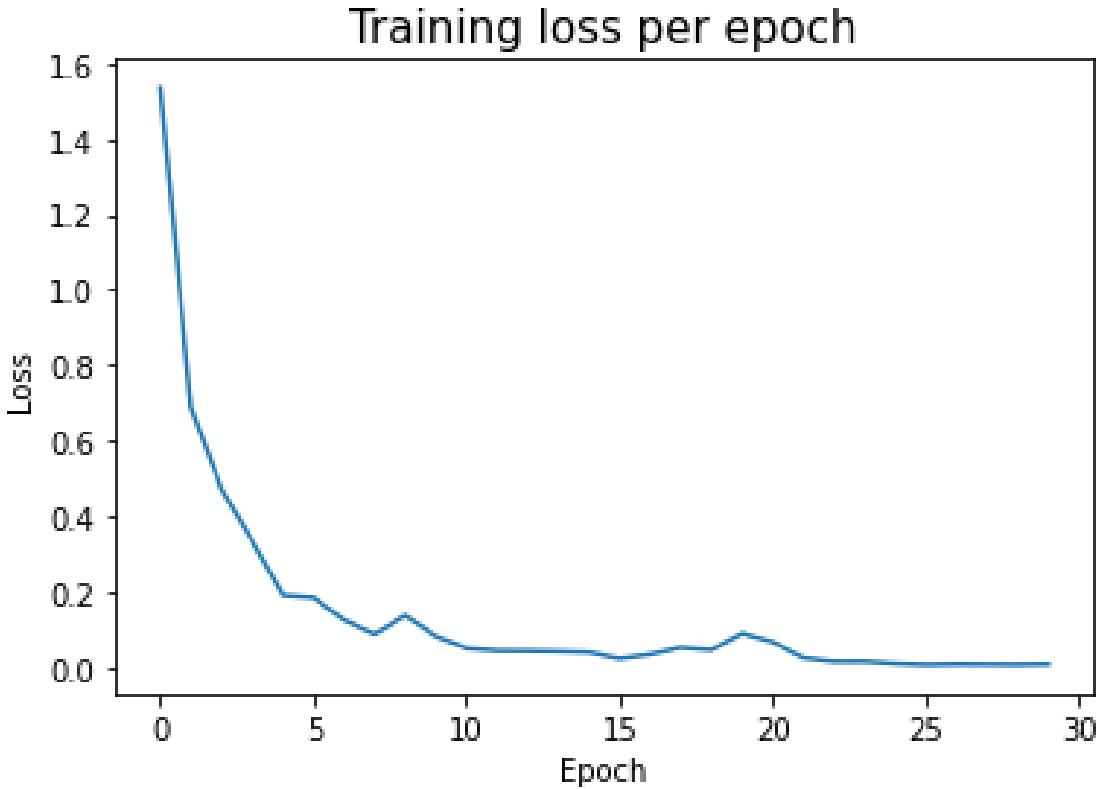
Epoch 27: Mean training loss per epoch: 0.007158982510143822

Epoch 28: Mean training loss per epoch: 0.006778088373485608

Epoch 29: Mean training loss per epoch: 0.007739723603137665

Γενικά τυπώσαμε το mean training loss γιατί έχει περισσότερη αξία και με βάση αυτό μελετάμε το μοντέλο μας.

Γραφικά έχουμε:



Παρατηρούμε πως το training loss εμφανίζει εκθετική μείωση: Στις πρώτες εποχές η βελτίωση που παρατηρείται είναι πολύ μεγάλη και απότομη. Καθώς όμως το πλήθος των εποχών αυξάνεται, το training loss τείνει να σταθεροποιηθεί. Επιπλέον, μπορούμε να παρατηρήσουμε πως γραφική παράσταση είναι μεν φθίνουσα, όχι όμως γνησιων, εμφανίζονται δηλαδή κάποια spikes στο training loss.

4. Έπειτα εκτυπώνουμε train και validation loss και κάνουμε αποτίμηση του μοντέλου στο validation αλλά και test set. Ενδεικτικά έχουμε:

Epoch 0: Mean training loss per epoch: 1.5056708128363998

Epoch 0: Mean validation loss per epoch: 0.7791859980891732

Epoch 1: Mean training loss per epoch: 0.654705160414731

Epoch 1: Mean validation loss per epoch: 0.48087314516305923

Epoch 2: Mean training loss per epoch: 0.3870095723757037

Epoch 2: Mean validation loss per epoch: 0.3042486796484274

Epoch 3: Mean training loss per epoch: 0.35987219242034135

Epoch 3: Mean validation loss per epoch: 0.22467734434587114

Epoch 4: Mean training loss per epoch: 0.20621883233112318

Epoch 4: Mean validation loss per epoch: 0.2115059372256784

Epoch 5: Mean training loss per epoch: 0.1767010845657852

Epoch 5: Mean validation loss per epoch: 0.1368019871194573

Epoch 6: Mean training loss per epoch: 0.12703017719365933

Epoch 6: Mean validation loss per epoch: 0.10819845520617331

⋮

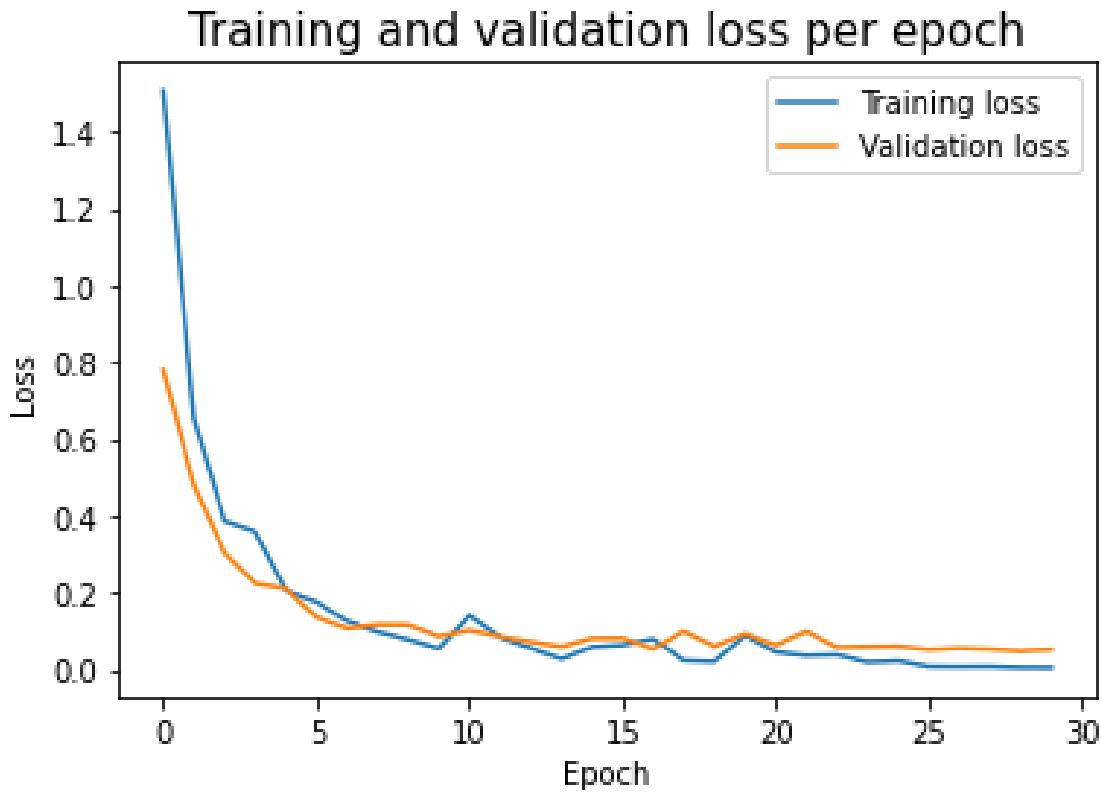
Epoch 28: Mean training loss per epoch: 0.005907064901354412

Epoch 28: Mean validation loss per epoch: 0.049359066922025865

Epoch 29: Mean training loss per epoch: 0.005565057737813159
 Epoch 29: Mean validation loss per epoch: 0.05312342628638041

Το μοντέλο έχει accuracy στα validation data ίση με 98,88 %

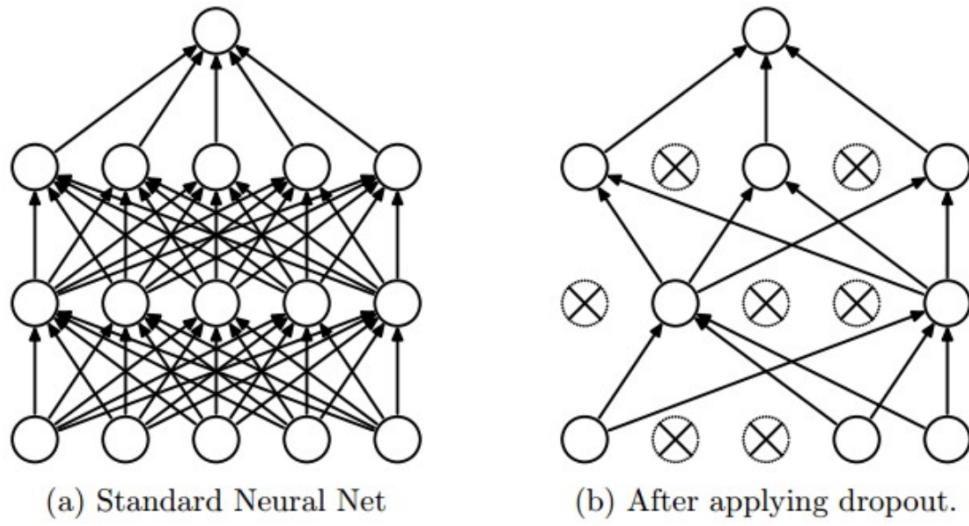
Γραφικά έχουμε:



Από το παραπάνω διάγραμμα βλέπουμε ότι το validation loss φθήνει λιγότερο γρήγορα από το training και εμφανίζει αρκετά περισσότερα spikes. Διαπιστώνουμε επίσης ότι καθώς αυξάνεται το πλήθος εποχών το validation loss παραμένει σταθετά ψηλότερο από το training loss, το οποίο είναι λογικό, καθώς τα δεδομένα εκπαίδευσης το μοντέλο μας τα βλέπει πολλαπλές φορές και προσαρμόζεται με βάση αυτά, ενώ τα validation δεδομένα είναι δεδομένα που δεν επιδρούν στην εκπαίδευση του μοντέλου.

5. Έπειτα προσθέτουμε στο μοντέλο μας Dropout και L2 Regularization. Αναλύοντας συνοπτικά την κάθε μέθοδο:

- Dropout: Μέθοδος αντιμετώπισης του overfitting η οποία βασίζεται στην τυχαία διαγραφή κόμβων του νευρωνικού. Κατά τη διάρκεια της εκπαίδευσης, μερικοί κόμβοι κάποιων επιπέδων αγνοούνται τυχαία. Αυτό έχει σαν αποτέλεσμα το αντίστοιχο επίπεδο του νευρωνικού να έχει διαφορετικό αριθμό κόμβων και διαφορετική σύνδεση με το προηγούμενο επίπεδο. Ένα παράδειγμα φαίνεται παρακάτω:



- L2 Regularization: Μέθοδος αντιμετώπισης του overfitting, η οποία αποδίδει μεγαλύτερο κόστος στο loss function σε μοντέλα που χρησιμοποιούν μεγάλες τιμές στα βάρη του νευρωνικού. Στόχος είναι τελικά το μοντέλο που θα προκύψει να έχει όσο το δυνατόν πιο μικρά βάρη.

Δοκιμάζουμε διαφορετικές τιμές για dropout και L2 regularization και για κάθε τιμή εκτινάχουμε τα train και validation loss και κάνουμε αποτίμηση του μοντέλου στο validation αλλά και test set.

- Για dropout=0.2 και weight_decay=0.00001 έχουμε:

```

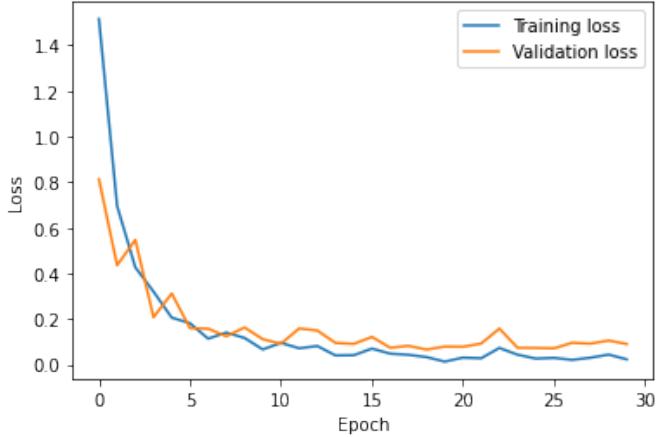
Epoch 0: Mean training loss per epoch: 1.5158054947853088
Epoch 0: Mean validation loss per epoch: 0.8129857515587526
Epoch 1: Mean training loss per epoch: 0.6956422830069506
Epoch 1: Mean validation loss per epoch: 0.43558233625748577
Epoch 2: Mean training loss per epoch: 0.426741097205215
Epoch 2: Mean validation loss per epoch: 0.5470025109894135
Epoch 3: Mean training loss per epoch: 0.3202280919033068
Epoch 3: Mean validation loss per epoch: 0.2079173872137771
Epoch 4: Mean training loss per epoch: 0.20605166387502794
Epoch 4: Mean validation loss per epoch: 0.3122241474030649
Epoch 5: Mean training loss per epoch: 0.18168536532256338
Epoch 5: Mean validation loss per epoch: 0.1597447676638908
:
Epoch 28: Mean training loss per epoch: 0.04401633861405706
Epoch 28: Mean validation loss per epoch: 0.10559039366061744
Epoch 29: Mean training loss per epoch: 0.02321734309101615
Epoch 29: Mean validation loss per epoch: 0.09046736797180903

```

Το μοντέλο έχει accuracy στα validation data ίση με 97,40 %

Γραφικά έχουμε:

Training and validation loss per epoch, with Dropout=0.2 and L2 Regularization with weight decay=1e-05



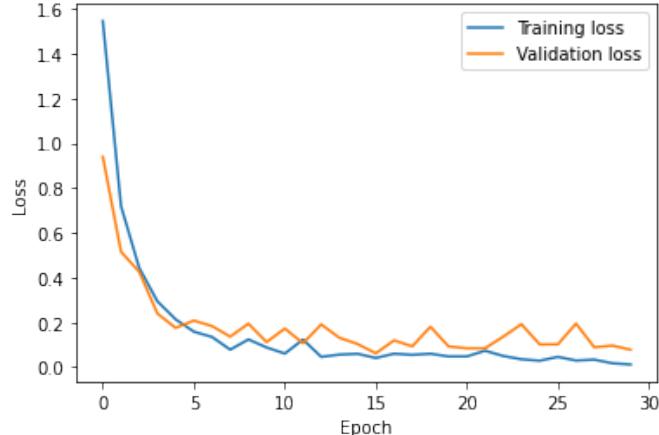
- Για τις επόμενες περιπτώσεις παραθέτουμε τα αποτελέσματα στο accuracy και γραφικά:

- Για dropout=0.4 και weight_decay=0.00001 έχουμε:

To μοντέλο έχει accuracy στα validation data ίση με 97,96 %

Γραφικά έχουμε:

Training and validation loss per epoch, with Dropout=0.4 and L2 Regularization with weight decay=1e-05

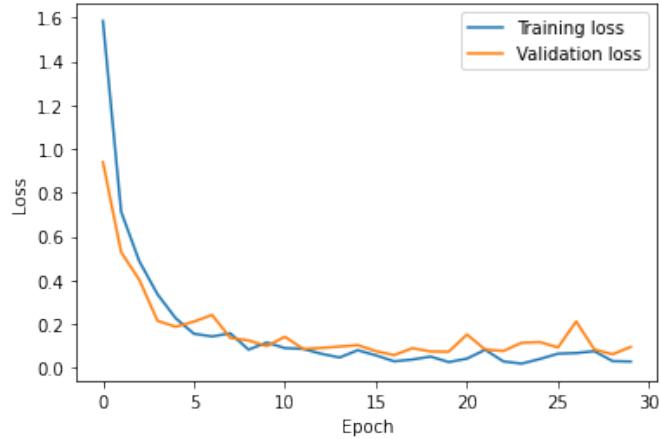


- Για dropout=0.6 και weight_decay=0.00001 έχουμε:

To μοντέλο έχει accuracy στα validation data ίση με 97,2 %

Γραφικά έχουμε:

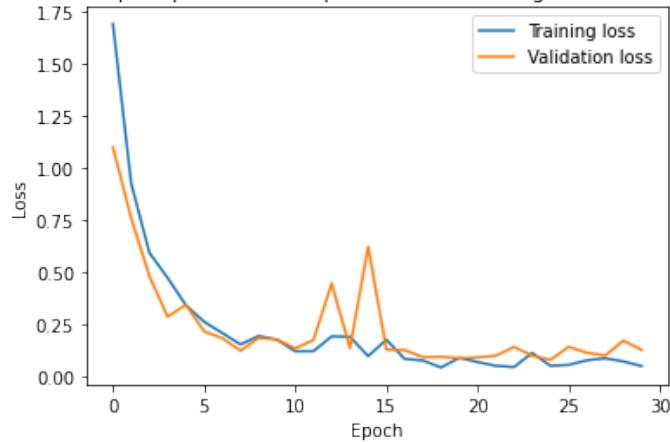
Training and validation loss per epoch, with Dropout=0.6 and L2 Regularization with weight decay=1e-05



- Για dropout=0.2 και weight_decay=0.001 έχουμε:

Το μοντέλο έχει accuracy στα validation data ίση με 88,34 % Γραφικά έχουμε:

Training and validation loss per epoch, with Dropout=0.2 and L2 Regularization with weight decay=0.001

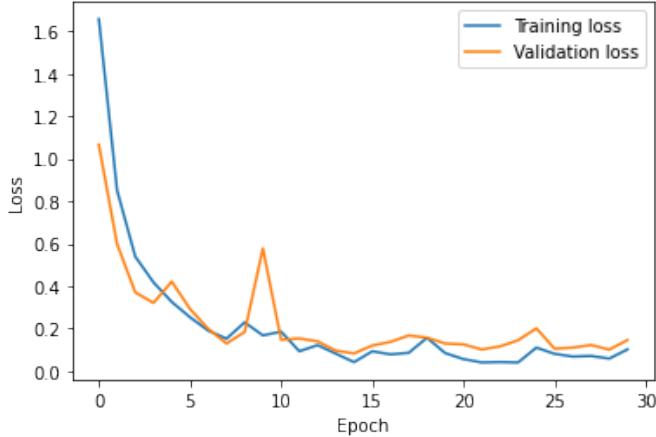


- Για dropout=0.4 και weight_decay=0.001 έχουμε:

Το μοντέλο έχει accuracy στα validation data ίση με 95,92 %

Γραφικά έχουμε:

Training and validation loss per epoch, with Dropout=0.4 and L2 Regularization with weight decay=0.001

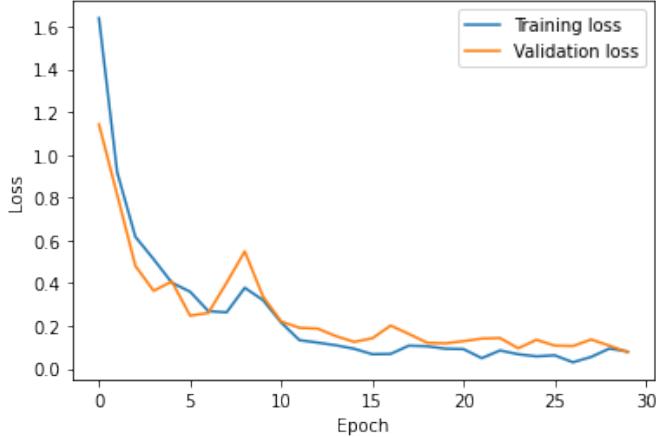


- Για dropout=0.6 και weight_decay=0.001 έχουμε:

Το μοντέλο έχει accuracy στα validation data ίση με 97,77 %

Γραφικά έχουμε:

Training and validation loss per epoch, with Dropout=0.6 and L2 Regularization with weight decay=0.001



Από τα παραπάνω διαπιστώνουμε πως οι βέλτιστες τιμές dropout και L2 Regularization για το μοντέλο μας είναι 0.4 και 0.00001 αντίστοιχα.

Γενικά μπορούμε να σχολιάσουμε πως οι μέθοδοι dropout και L2 Regularization, για τις τιμές 0.4 και 0.00001 αντίστοιχα, δεν βοήθησαν στην βελτίσωση της απόδοσης του μοντέλου μας (μείωση της ακρίβειας από 98.88% σε 97.2%). Γενικά αυτό οφείλεται πιθανόν στο ότι έχουμε λίγα δεδομένα στο dataset, συνεπώς δεν έχουμε overfitting. Μπορούμε επίσης να παρατηρήσουμε ότι για τις τιμές αυτές η γραφική παράσταση του validation loss δεν είναι τόσο ομαλή καθώς εμφανίζει λίγο πιο πολλά spikes.

6. Στο ερώτημα αυτό υλοποιούμε Early Stopping και Checkpoints. To Early Stopping έχει ως στόχο να σταματήσει την εκπαίδευση του νευρωνικού πριν αρχίσει να κάνει overfit στα train δεδομένα, επιτρέποντας παράλληλα σε εμάς να ορίζουμε έναν αυθαίρετα μεγάλο αριθμό εποχών για την εκπαίδευση του μοντέλου μας και αποδεσμεύοντάς μας από την ανάγκη να αναζητούμε τον βέλτιστο αριθμό εποχών χειροκίνητα, με δοκιμές. Τα Checkpoints συμβάλλουν στη διάτηρηση του καλύτερου μοντέλου σε περίπτωση εμφάνισης κάποιας βλάβης στο σύστημά μας και διακοπής του κώδικά μας: Η εκπαίδευση ενός νευρωνικού είναι μια διαδικασία η οποία απαιτεί

πολύ χρόνο. Για τον λόγο αυτό είναι ιδιαίτερα χρήσιμο κατά την εκπαίδευση να διατηρούμε αντίγραφα του καλύτερου κάθε φορά μοντέλου, ώστε να μην χρειάζεται να ξεκινήσουμε την εκπαίδευση από την αρχή σε περίπτωση κάποιας βλάβης, κάτι το οποίο θα είχε μεγάλο κόστος.

Σημειώνουμε ότι κατά την υλοποίηση του early stopping μπορούμε να ορίσουμε μία παράμετρο patience, η οποία ουσιαστικά ορίζει σε πόσες διαδοχικές εποχές πρέπει το μοντέλο μας να μην έχει παρουσιάσει βελτίωση στο loss πριν σταματήσουμε την εκπαίδευση. Ορίζουμε μετά από πειραματισμούς την τιμή αυτή ίση με 4.

Εκτυπώνουμε τα train και validation loss και κάνουμε αποτίμηση του μοντέλου στο validation αλλά και test set:

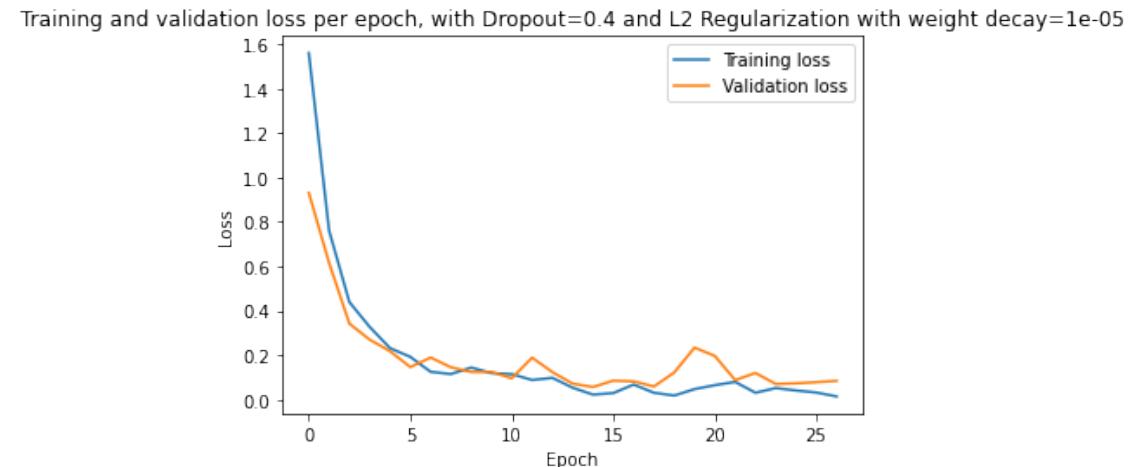
```

Epoch 0: Mean training loss per epoch: 1.5597727303151732
Epoch 0: Mean validation loss per epoch: 0.9309323088211172
Epoch 1: Mean training loss per epoch: 0.7607246707986902
Epoch 1: Mean validation loss per epoch: 0.6155104913255748
Epoch 2: Mean training loss per epoch: 0.4391309524023974
Epoch 2: Mean validation loss per epoch: 0.34224016920608635
Epoch 3: Mean training loss per epoch: 0.32747790543017563
Epoch 3: Mean validation loss per epoch: 0.2702183944775778
:
Epoch 25: Mean training loss per epoch: 0.032539696655132705
Epoch 25: Mean validation loss per epoch: 0.07891007047488957
Number of times validation loss has not decreased: 2
Epoch 26: Mean training loss per epoch: 0.014798500523816243
Epoch 26: Mean validation loss per epoch: 0.08501042661887101
Number of times validation loss has not decreased: 3 Early stopping...

```

Το μοντέλο έχει accuracy στα validation data ίση με 97,96 %

Γραφικά έχουμε:



Διαπιστώνουμε πως η εκπαίδευση του μοντέλου μας σταματά τώρα στις 26 εποχές. Παρατηρούμε βέβαια πως το μοντέλο μας τώρα, αν και έχει ακόμα υψηλή απόδοση (ακρίβεια 97.96%) δεν έχει φτάσει την μέγιστη απόδοση (98.88%) που πετύχαμε νωρίτερα.

7. Στο βήμα αυτό εξετάζουμε την περίπτωση του bidirectional LSTM cell. Η διαφορά του bidirectional LSTM με το απλό LSTM είναι ότι στο πρώτο έχουμε ουσιαστικά 2 LSTM layer cells, το ένα για την πορεία από το παρελθόν στο μέλλον και το άλλο από το μέλλον στο παρελθόν.

Ουσιαστικά ενώνονται και νευρώνες ανάποδης κατεύθυνσης με ίδια έξοδο. Έτσι μπορεί η έξοσης να λάβει μηνύματα και από τα δύο προηγούμενα hidden layers. Αυτό αυξάνει την ποσότητα πληροφορίας στο δίκτυο.

Εκτυπώνουμε τα train και validation loss και κάνουμε αποτίμηση του μοντέλου στο validation αλλά και test set.

```
Epoch 0: Mean training loss per epoch: 1.0488782431240435
Epoch 0: Mean validation loss per epoch: 0.5595324166557368
Epoch 1: Mean training loss per epoch: 0.3304234399149815
Epoch 1: Mean validation loss per epoch: 0.2572041162673165
Epoch 2: Mean training loss per epoch: 0.19569601550422333
Epoch 2: Mean validation loss per epoch: 0.16673769703244462
Epoch 3: Mean training loss per epoch: 0.11433658648834184
Epoch 3: Mean validation loss per epoch: 0.08472923424971454
```

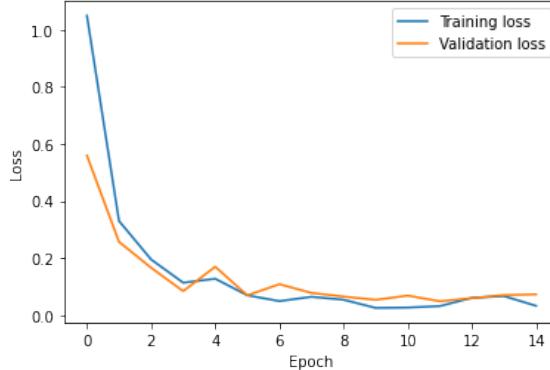
:

```
Epoch 13: Mean training loss per epoch: 0.06734978977797759
Epoch 13: Mean validation loss per epoch: 0.07096118582686518
Number of times validation loss has not decreased: 2
```

```
Epoch 14: Mean training loss per epoch: 0.03328649344664343
Epoch 14: Mean validation loss per epoch: 0.0734513769771986
Number of times validation loss has not decreased: 3
Early stopping...
```

To μοντέλο έχει accuracy στα validation data ίση με 97,77 %

Bidirectional LSTM training and validation loss per epoch, with Dropout=0.4 and L2 Regularization with weight decay=1e-05



Διαπιστώνουμε πως το bidirectional μοντέλο χρειάζεται πολύ λιγότερες εποχές για την εκπαίδευσή του (σταματά με early stopping ήδη στις 14 εποχές), αλλά παρόλαυτα δεν επιτυγχάνει την μέγιστη ακρίβεια (98.88%) σε σύγκριση με όσα μοντέλα δοκιμάσαμε νωρίτερα. Επιπλέον, οι γραφικές παραστάσεις του training και του validation loss είναι πολύ πιο ομαλές.

Καλύτερο Μοντέλο

Το καλύτερο μοντέλο που προέκυψε είναι το απλό μοντέλο. Για αυτό το μοντέλο χρησιμοποιήσαμε αριθμό εποχών 50, dropout =0.4, weight_decay=0.00001 , και Early Stopping και προέκυψαν τα εξής αποτελέσματα:

```
Epoch 0: Mean training loss per epoch: 1.5037664338394447
Epoch 0: Mean validation loss per epoch: 0.9613376505234662
```

```
Epoch 1: Mean training loss per epoch: 0.7034142728205081
Epoch 1: Mean validation loss per epoch: 0.5375091556240531
```

Epoch 2: Mean training loss per epoch: 0.4137639021431958
Epoch 2: Mean validation loss per epoch: 0.48981861726326104

Epoch 3: Mean training loss per epoch: 0.294411307628508
Epoch 3: Mean validation loss per epoch: 0.3128393489648314

⋮

Epoch 40: Mean training loss per epoch: 0.009413504500709543
Epoch 40: Mean validation loss per epoch: 0.09631255214523715
Number of times validation loss has not decreased: 2

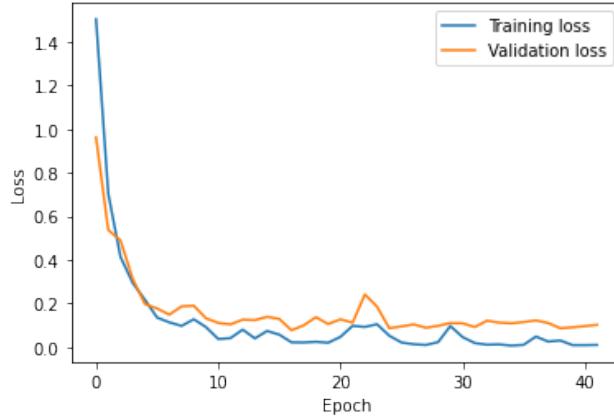
Epoch 41: Mean training loss per epoch: 0.01027332686245683
Epoch 41: Mean validation loss per epoch: 0.10248222800360321
Number of times validation loss has not decreased: 3

Early stopping...

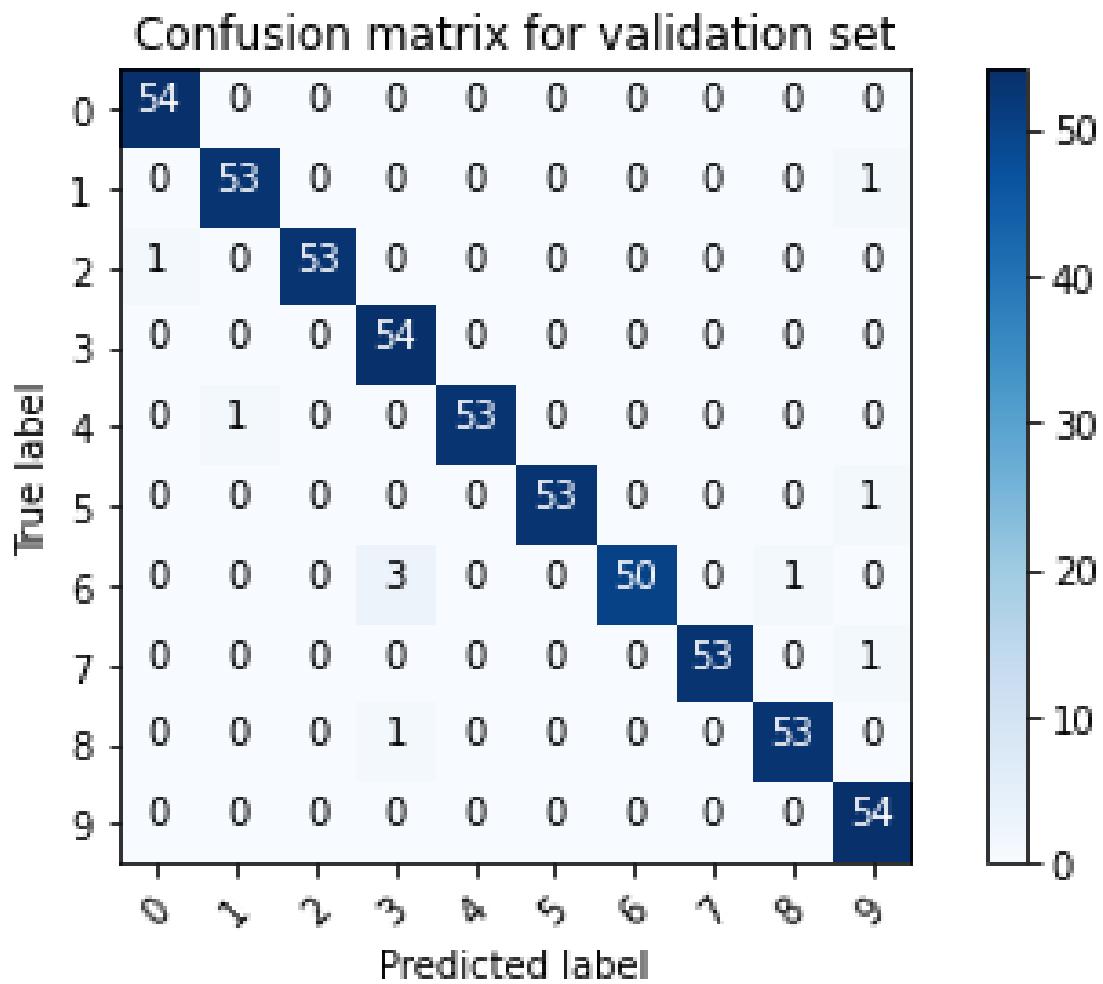
Διαπιστώνουμε πως η εκπαίδευση του μοντέλου μας σταματά στις 41 εποχές. Το μοντέλο έχει accuracy στα validation data ίση με 98,14 % και 99,33 % στα test data

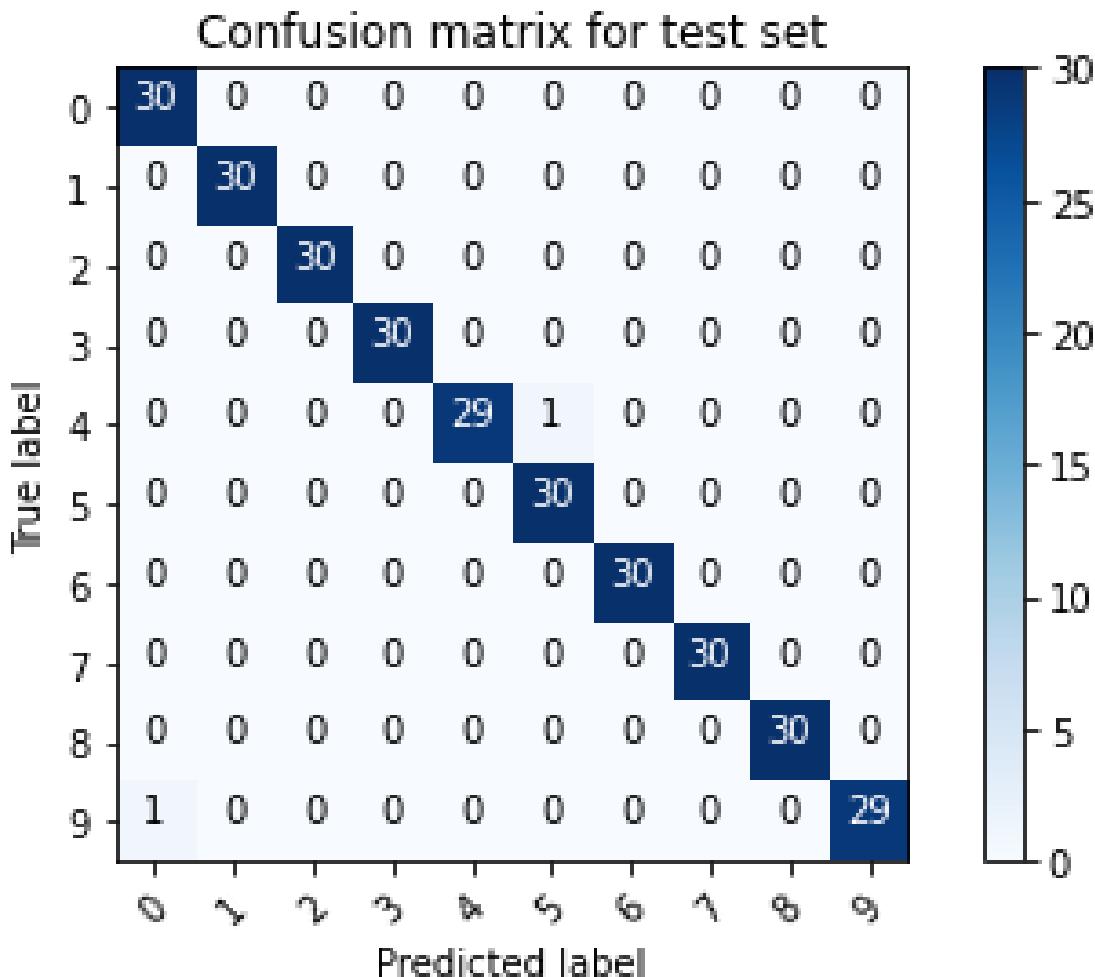
Γραφικά έχουμε:

LSTM training and validation loss per epoch, with Dropout=0.4 and L2 Regularization with weight decay=1e-05



Παρατηρούμε ότι οι γραφικές παραστάσεις του training και του validation loss είναι αρκετά ομαλές.
Παρακάτω βλέπουμε το confusion matrix για το validation και το test set





Παρατηρούμε ότι στο validation set προκύπτουν ελάχιστα λάθη στις κλάσεις 2, 4, 6 ενώ στο test set προκύπτουν ακόμα λιγότερα στην κλάση 9. Επομένως, μπορούμε να πούμε πως τα λάθη είναι σχετικά "τυχαία", δηλαδή το μοντέλο μας δεν εμφανίζει ιδιαίτερη τάση να κάνει ένα συγκεκριμένο λάθος και δεν είναι biased. Ακόμα και στις κλάσεις πάντως που παρουσιάζονται λάθη, αυτά είναι αρκετά σπάνια, το οποίο είναι αναμενόμενο, δεδομένου ότι έχουμε accuracy 99.33 .