Αναγνώριση Προτύπων
2η Εργαστηριακή Άσκηση

Ομάδα εργασίας :

Συμπέθερος Αριστοτέλης-Γεώργιος (ΑΜ :031 16005)

Σιφναίος Σάββας (ΑΜ : 031 16080)

Εξάμηνο: 9ο (Προπτυχιακό) **Σχολή:** ΗΜΜΥ

Εισαγωγή:

Στα πλαίσια της δεύτερης εργαστηριακής άσκησης θα ασχοληθούμε με την αναγνώριση ακουστικών σημάτων. Συγκεκριμένα, στην διάθεσή μας έχουμε εκφωνήσεις ψηφίων από το 0 έως και το 9 και σκοπός μας είναι να αναπτύξουμε ένα σύστημα ικανό να αναγνωρίσει και στην συνέχεια να ταξινομήσει τα ακουστικά αυτά ψηφία στην σωστή κατηγορία.

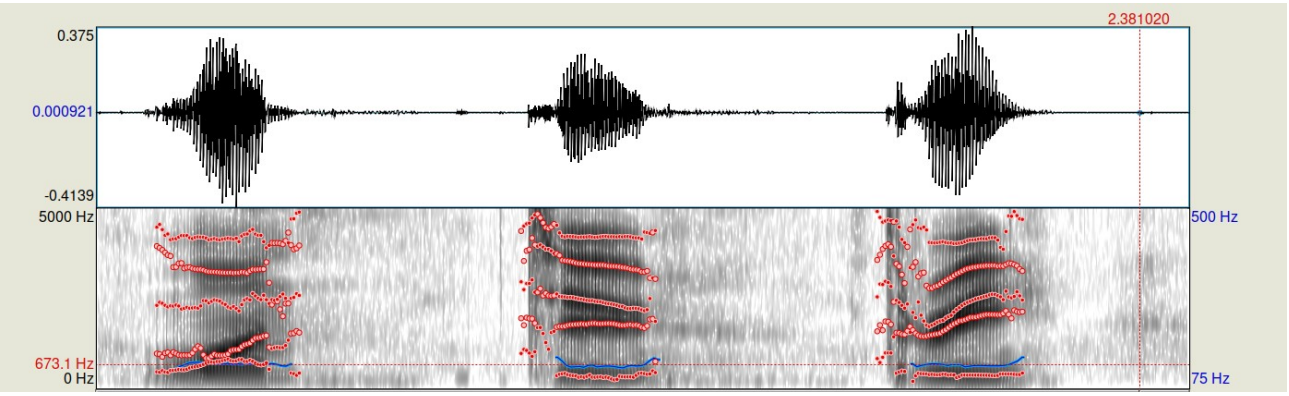
Προπαρασκευαστικό Μέρος

Στο προπαρασκευαστικό μέρος της συγκεκριμένης άσκησης, θα επικεντρωθούμε στην εξαγωγή χαρακτηριστικών από ηχητικά σήματα, τα οποία χαρακτηριστικά στην συνέχεια θα χρησιμοποιηθούν ως συμπαγείς αναπαραστάσεις του αρχικού ηχητικού μας σήματος,ενώ στο τέλος του μέρους αυτού θα επιχειρήσουμε να λύσουμε ένα απλό πρόβλημα προβλέψεων με την χρήση αναδρομικών νευρωνικών δικτύων.

Βήμα 1:

Αρχικά, εξετάζουμε τα ηχητικά αρχεία onetwothree1.wav και onetwothree8.wav , τα οποία περιέχουν τις εκφωνήσεις των ψηφίων ένα έως και τρία από έναν άντρα ομιλητή και από μία γυναίκα αντίστοιχα.

Αναφορικά με τον πρώτο αρχείο ήχου έχουμε :



Εικόνα 1. *Συνολικό σήμα ήχου ομιλητή 1.*

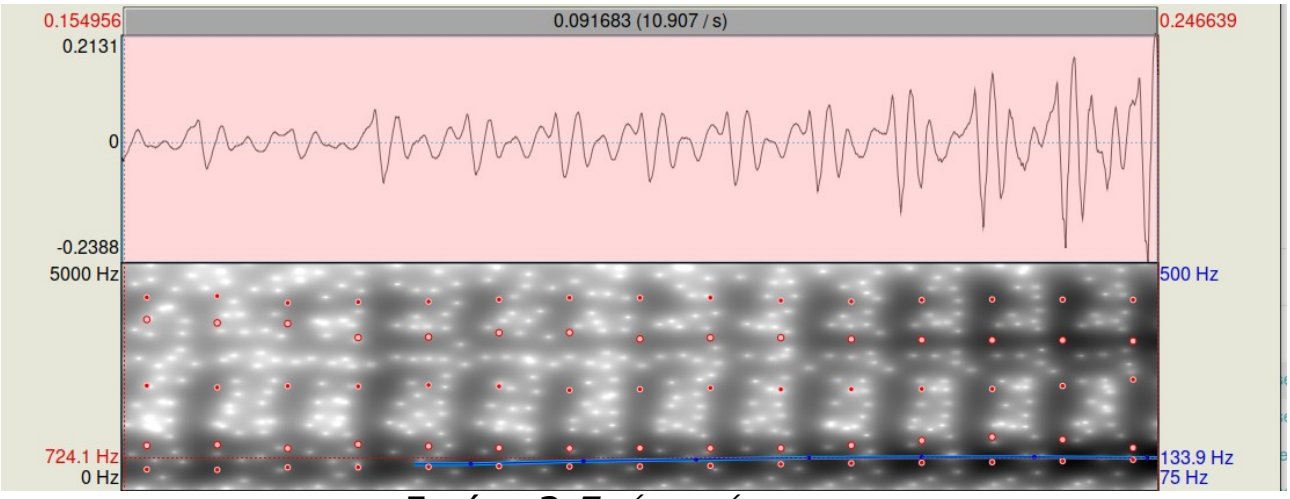
- Για το πρώτο "ου" (στο one) :
  - Το pitch είναι 133.9 Hz
  - 1ο Formant 545.5 Hz
  - 2ο Formant 953.7 Hz
  - 3ο Formant 2255 Hz

- Για το "α" :
  - Το pitch είναι 134.7 Hz
  - 1ο Formant 712.4 Hz
  - 2ο Formant 1094.1 Hz
  - 3ο Formant 2213.3 Hz

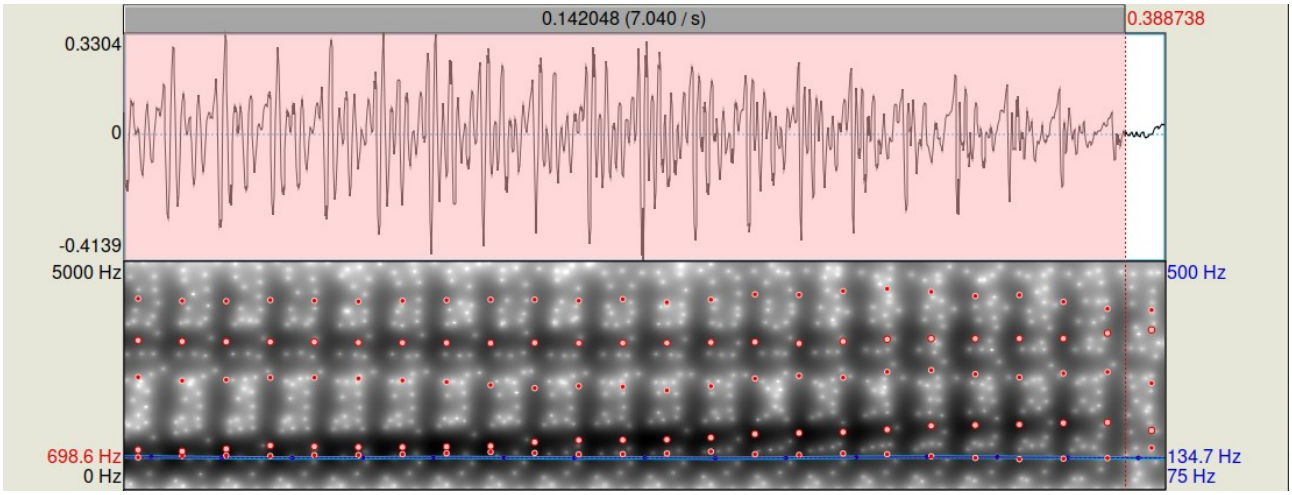
- Για το δεύτερο "ου" :
  - Το pitch είναι 129.2 Hz
  - 1ο Formant 400.5 Hz
  - 2ο Formant 1761.8 Hz
  - 3ο Formant 2353.4 HZ

- Για το "ε" :
  - Το pitch είναι 131.2 Hz
  - 1ο Formant 402.9 Hz
  - 2ο Formant 1862.3 Hz
  - 3ο Formant 2294.0 Hz

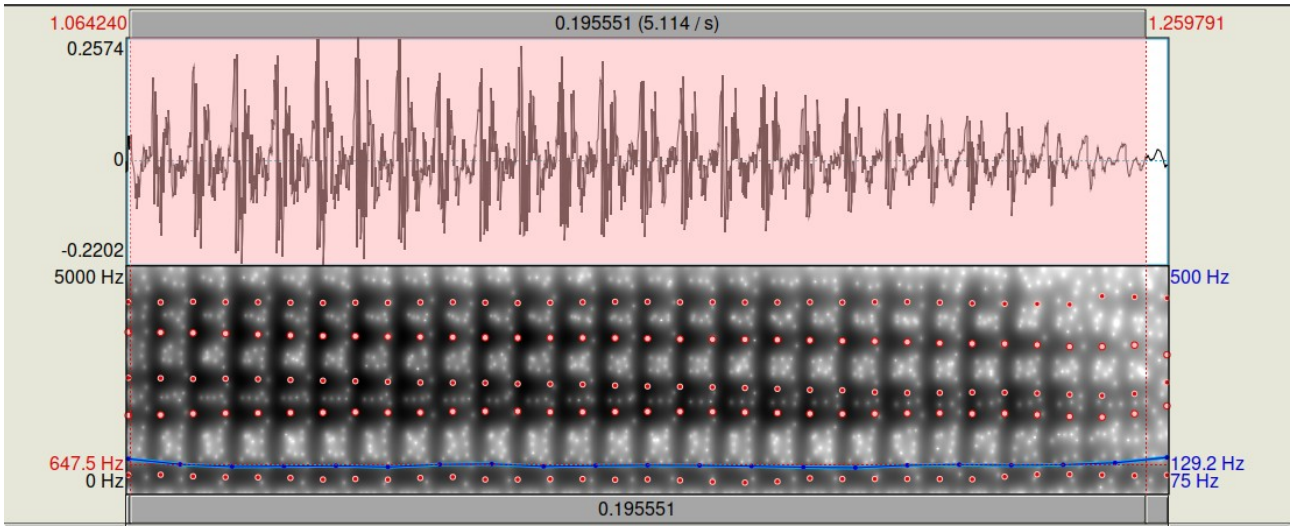
Τα παραπάνω επιβεβαιώνονται με τα εξής στιγμιότυπα από το πρόγραμμα Praat



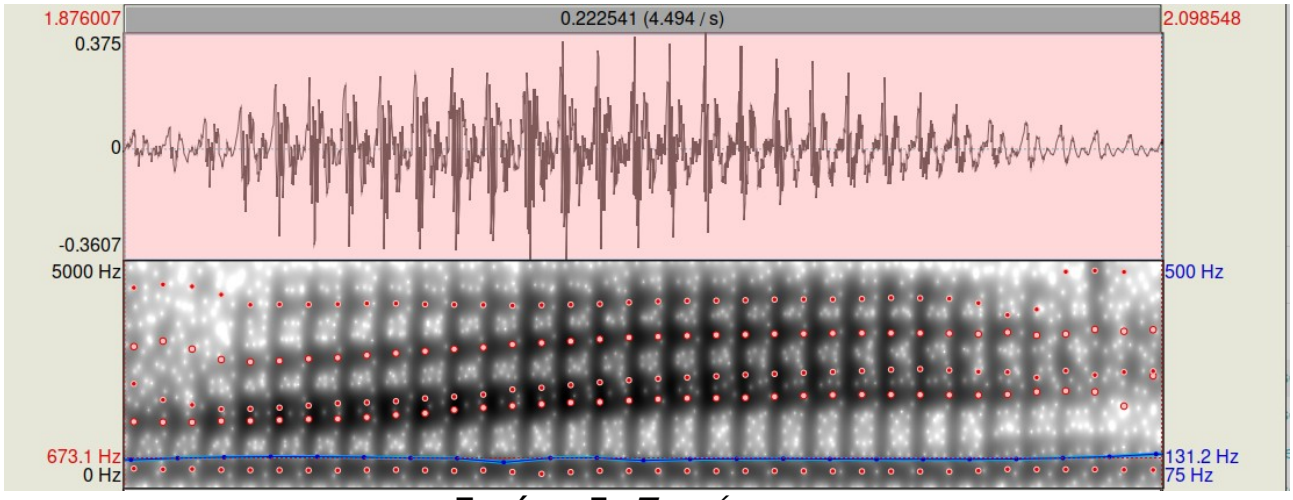
Εικόνα 2.*Πρώτο φώνημα ου.*



Εικόνα 3. *Το φώνημα α.*

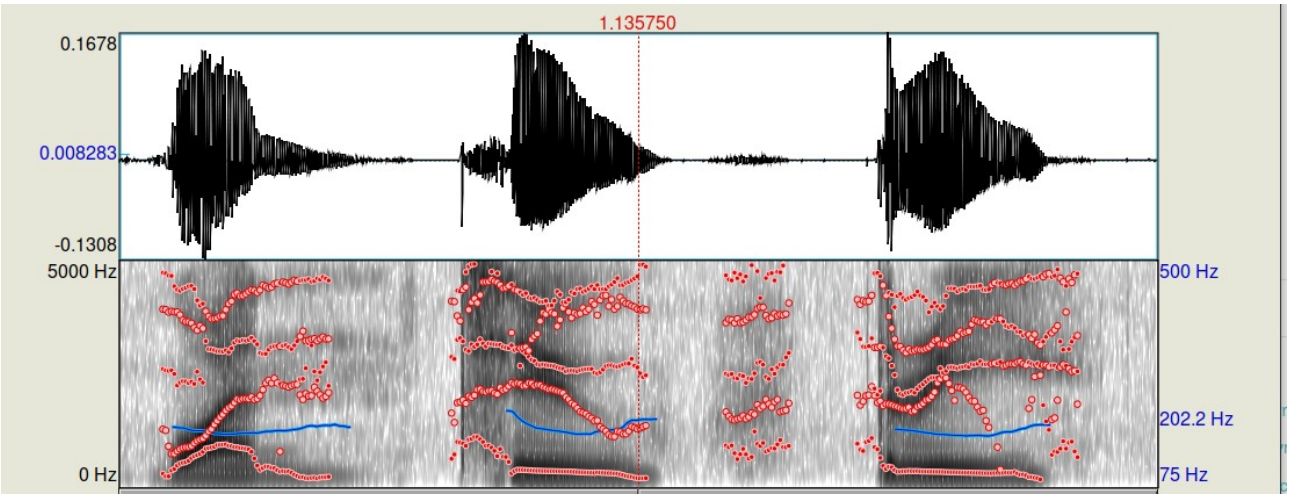


Εικόνα 4. *Δεύτερο φώνημα ου*



Εικόνα 5. *Το φώνημα ε*

Αναφορικά με τον δεύτερο αρχείο ήχου έχουμε :



Εικόνα 6. *Συνολικό σήμα ήχου ομιλητή 8.*

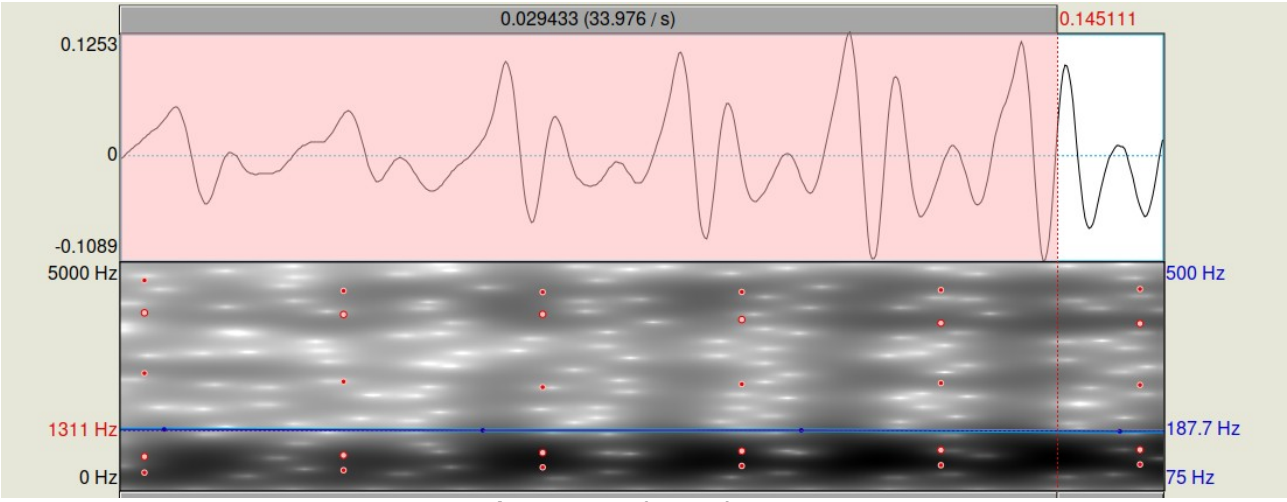
- Για το πρώτο "ου" (στο one) :
  - Το pitch είναι 187.8 Hz
  - 1ο Formant 596.5 Hz
  - 2ο Formant 979.2 Hz
  - 3ο Formant 2383 Hz

- Για το "α" :
  - Το pitch είναι 177.7 Hz
  - 1ο Formant 797.8 Hz
  - 2ο Formant 1416.6 Hz
  - 3ο Formant 2842 Hz

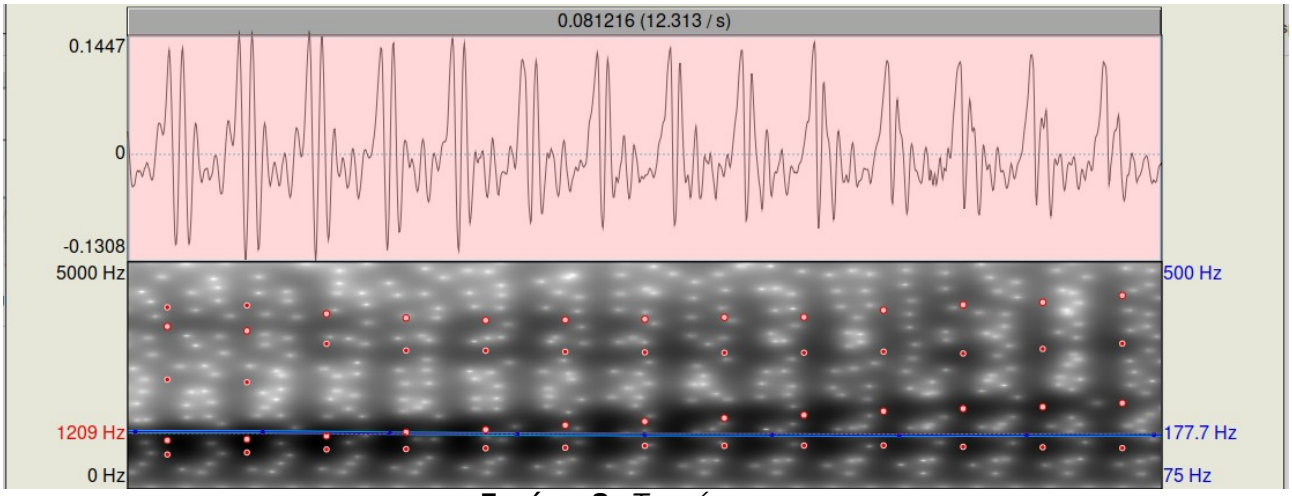
- Για το δεύτερο "ου" :
  - Το pitch είναι 186.8 Hz
  - 1ο Formant 332.8 Hz
  - 2ο Formant 1740.4 Hz
  - 3ο Formant 2683.2 HZ

- Για το "ε" :
  - Το pitch είναι 178.1 Hz
  - 1ο Formant 400.7 Hz
  - 2ο Formant 2137.0 Hz
  - 3ο Formant 2792.4 Hz

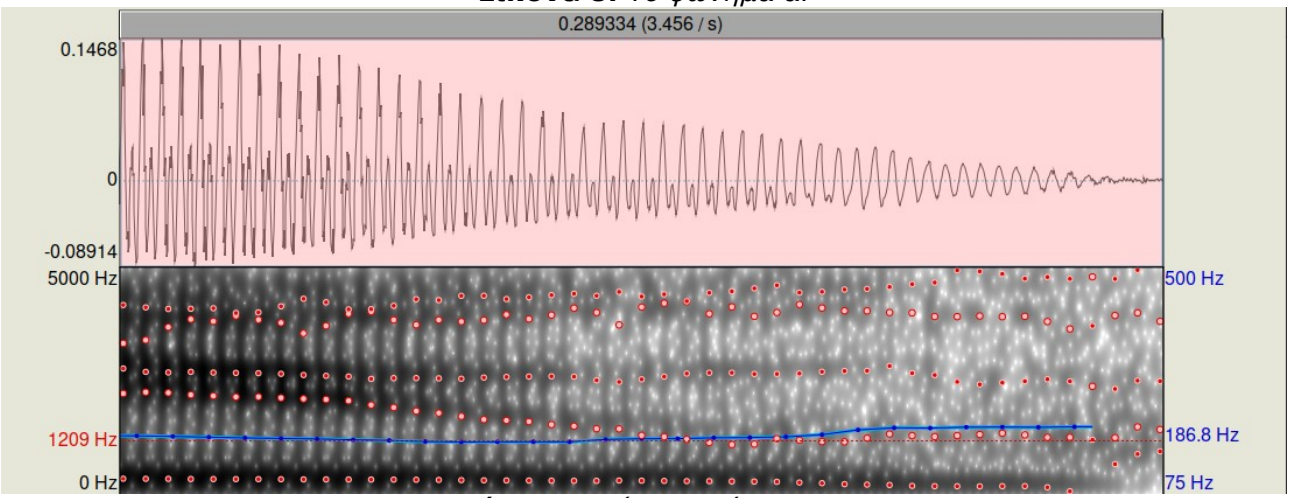
Τα παραπάνω επιβεβαιώνονται με τα εξής στιγμιότυπα από το πρόγραμμα Praat



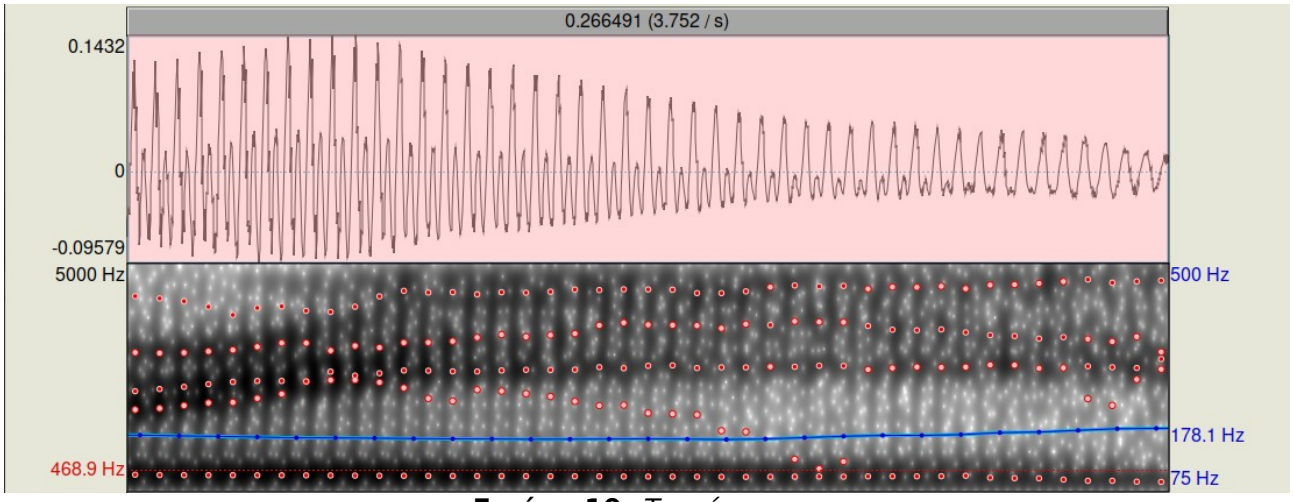
Εικόνα 7. *Πρώτο φώνημα ου.*



Εικόνα 8. *Το φώνημα α.*



Εικόνα 9. *Δεύτερο φώνημα ου.*



Εικόνα 10. *Το φώνημα ε.*

Βήμα 2:
Βασιζόμαστε στον parser.py που δίνεται για το βήμα 9 για την ανάγνωση το δεδομένων που περιέχονται στον τίτλο του κάθε wav αρχείου (ψηφίο και εκφωνητής) καθώς και για το διάβασμα των wav με το librosa.

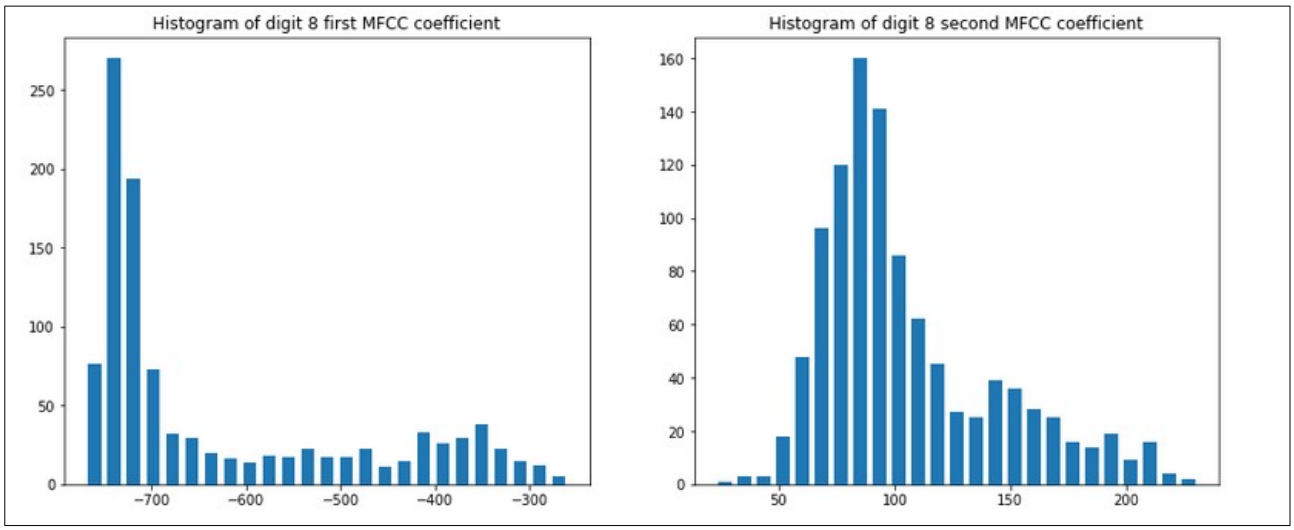
Βήμα 3:
Στην συνέχεια εξάγουμε τα Mel-Frequency Cepstral Coefficients (MFCCs)και συγκεκριμένα 13 χαρακτηριστικά ανά αρχείο με μήκος παραθύρου 25ms και βήμα 10ms. Τα παραπάνω έγιναν με την έτοιμη εντολή feature.mffc() της βιβλιοθήκης librosa και τις κατάλληλες παραμέτρους. Στην συνέχεια για τον υπολογισμό της πρώτης και δεύτερης παραγώγου των χαρακτηριστικών, δηλαδή της deltas και deltas-deltas κάνουμε χρήση της συνάρτησης feature.delta() (ομοίως της βιβλιοθήκης librosa) με την μόνη διαφορά ότι για την 2η παράγωγο δίνεται η επιπλέον παράμετρος order=2.

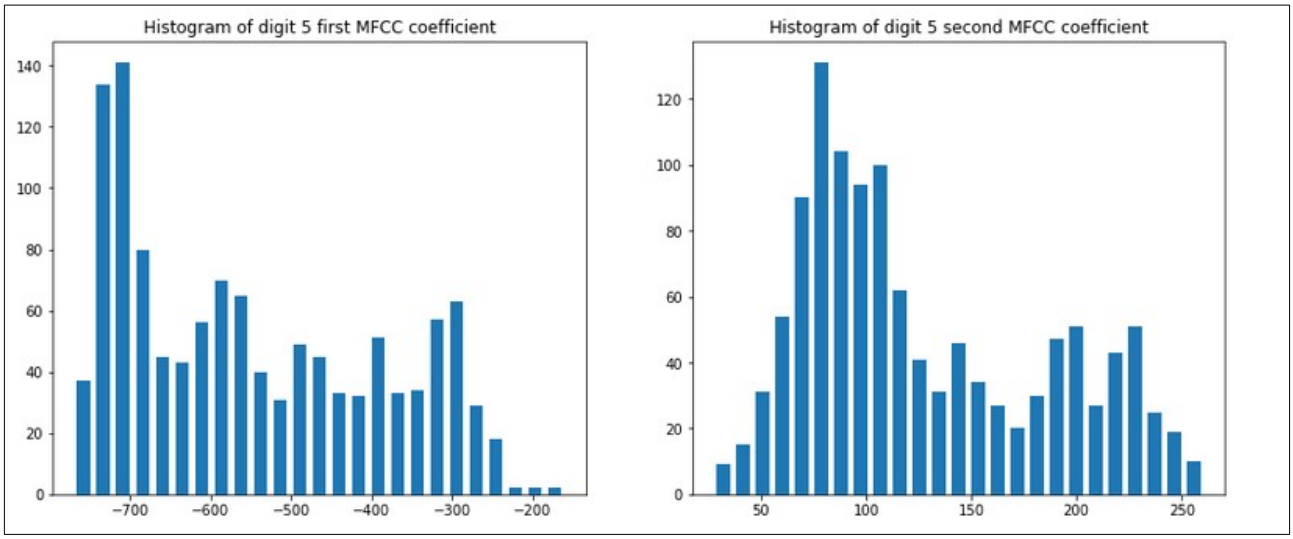
Στο σημείο αυτό, κρίνεται σκόπιμο να αναφέρουμε συνοπτικά την διαδικασία εξαγωγής των MFCCs. Αρχικά, πραγματοποιούμε τον χωρισμό του ηχητικού σήματος σε παράθυρα μήκους 25ms, τα κέντρα των οποίων απέχουν μεταξύ τους 10 ms. Το γεγονός ότι η απόσταση των κέντρων των παραθύρων είναι μικρότερη του μήκους αυτών συνεπάγεται την ύπαρξη επικαλύψεων του αρχικού σήματος από γειτονικά παράθυρα. Έχοντας χωρίσει το σήμα σε παράθυρα εφαρμόζουμε σε αυτά DFT προκειμένου να μεταφερθούμε στο πεδίο της συχνότητας, ώστε τελικά να λάβουμε τον λογάριθμο του φάσματος του ηχητικού σήματος. Στην συνέχεια, μετασχηματίζουμε το λογάριθμο τους φάσματος κάθε παραθύρου με βάση την κλίμακα MEL , η οποία είναι κλίμακα ευαισθησίας του ανθρώπινου αυτιού στις διαφορετικές συχνότητες και έχει προκύψει από ψυχο-ακουστικά πειράματα. Τέλος, στα μετασχηματισμένα λογαριθμικά φάσματα εφαρμόζουμε αντίστροφο DCT προκειμένου να επιστρέψουμε στον πεδίο του χρόνου και να καταλήξουμε με τα λεγόμενα MFCCs του σήματος.

Σε πρακτικές εφαρμογές, η παραπάνω αναλυθείσα διαδικασία πραγματοποιείται με συστοιχίες φίλτρων, οι οποίες καλούνται MEL Filterbanks , η ανάλυση της δομής των οποίων ξεφεύγει από τα πλαίσια της παρούσας εργασίας και για τον λόγο αυτό δεν παρατίθεται.

Βήμα 4:

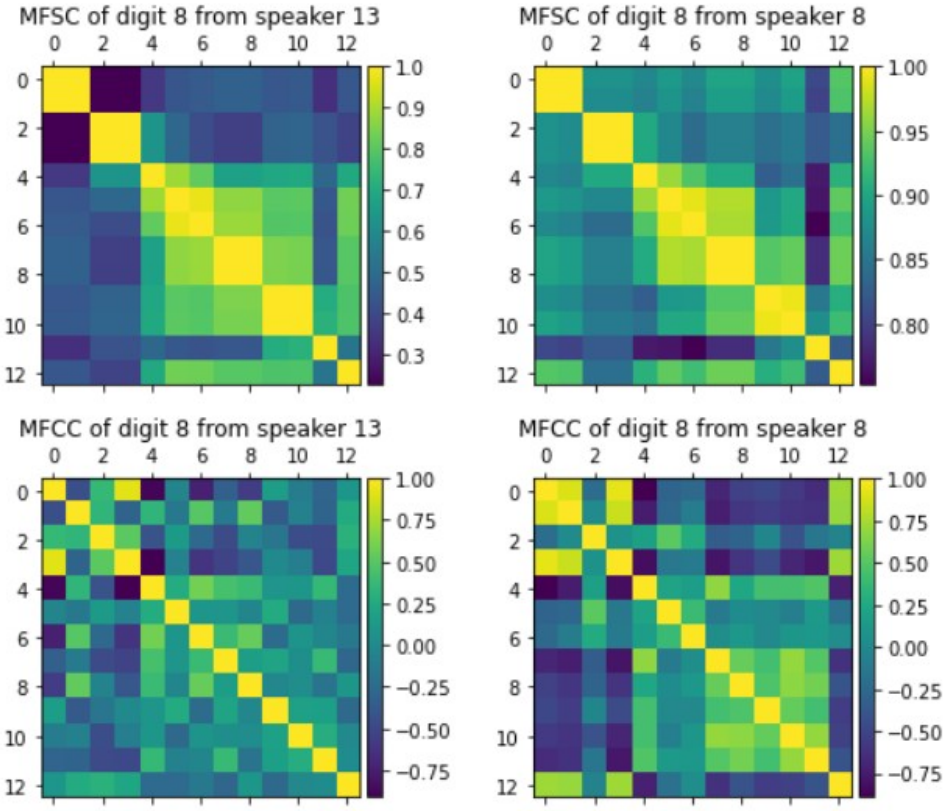
Αρχικά στο σημείο αυτό να αναφέρουμε πως δεδομένων των Α.Μ. μας , 16080 και 16005 οι μόνοι μη μηδενικοί αριθμοί που μπορούμε να λάβουμε είναι n1=8 και n2=5.
Στην συνέχεια απομονώνουμε όλες τις εκφωνήσεις του κάθε ψηφίου (των οποίων έχουμε ήδη υπολογίσει τα 13 MFCCs κάθε παραθύρου) και κατασκευάζουμε ένα ιστόγραμμα με βάση την 1η και 2η τιμή MFCC για κάθε παράθυρο κάθε εκφώνησης σε ένα ιστόγραμμα. Συνεπώς έχουμε 4 ιστογράμματα, 2 MFCCs (1ο και 2ο) για κάθε μία εκ των 2 εκφωνήσεων (n1 και n2). Τα ιστογράμματα αυτά φαίνονται παρακάτω.

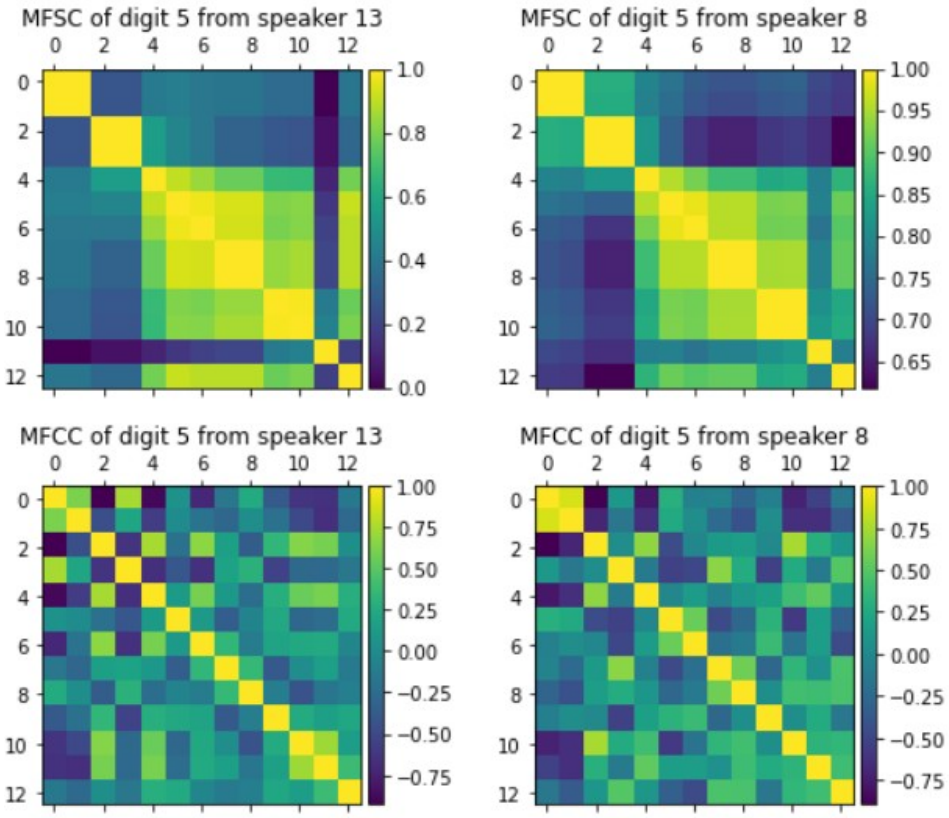




Για τα ιστογράμματα των 2 αριθμών βλέπουμε ότι για το ίδιο coefficient έχουμε μια σχετική ομοιότητα στη γενική μορφή, δηλαδή και για τα 2 ψηφία βλέπουμε για τον 1ο συντελεστή ότι οι τιμές είναι αρνητικές και περισσότερες βρίσκονται κοντά στο -700, από την άλλη για το 2ο συντελεστή έχουμε θετικές τιμές και έχουμε και στους 2 αριθμούς περισσότερες τιμές στην περιοχή 70-110. Ωστόσο υπάρχει σχετική απόκλιση.

Στην συνέχεια για 2εκφωνήσεις των n1 και n2 από 2 διαφορετικούς ομιλητές (τυχαία επιλέχθηκαν οι 8 και 13) , πέρα από τα MFCC που έχουμε ήδη εξάγει από προηγούμενα βήματα θα εξάγουμε (διαστάσεων 13 πάλι) και τα MEL Filterbank Spectral Coefficients (MFSCs) . Τα τελευταία εξάγονται αφού εφαρμοστεί η συστοιχία φίλτρων της κλίμακας Mel πάνω στο φάσμα του σήματος φωνής αλλά σε αντίθεση με τον υπολογισμό των MFCCs που εξηγήσαμε παραπάνω, δεν εφαρμόζουμε τον μετασχηματισμό DCT. Για τον υπολογισμό των παραπάνω κάνουμε χρήση των συναρτήσεων feuatue.melspectrogram και power_to_db της βιβλιοθήκης librosa. Όπου υπολογίζουμε πρώτα το ενέργεια του σήματος στην κλίμακα Mel με την πρώτη συνάρτηση και στην συνέχεια με την δεύτερη κάνουμε μετατροπή του σπεκτρογράμματος ενέργειας σε Decibel (dB) μονάδες. Στην συνέχεια και για τα 2 είδη χαρακτηριστικών (και για κάθε εκφώνηση) με χρήση της corrcoef της βιβλιοθήκης numpy του οποίου δίνουμε ως είσοδο έναν πίνακα μεγέθους 13x#_of_windows και επιστρέφεται ο πίνακας συσχέτισης μεταξύ των 13 χαρακτηριστικών (13x13) . Στην συνέχεια εμφανίζουμε τα παραπάνω στις εξής γραφικές (matplotlib.pyplot.matshow):



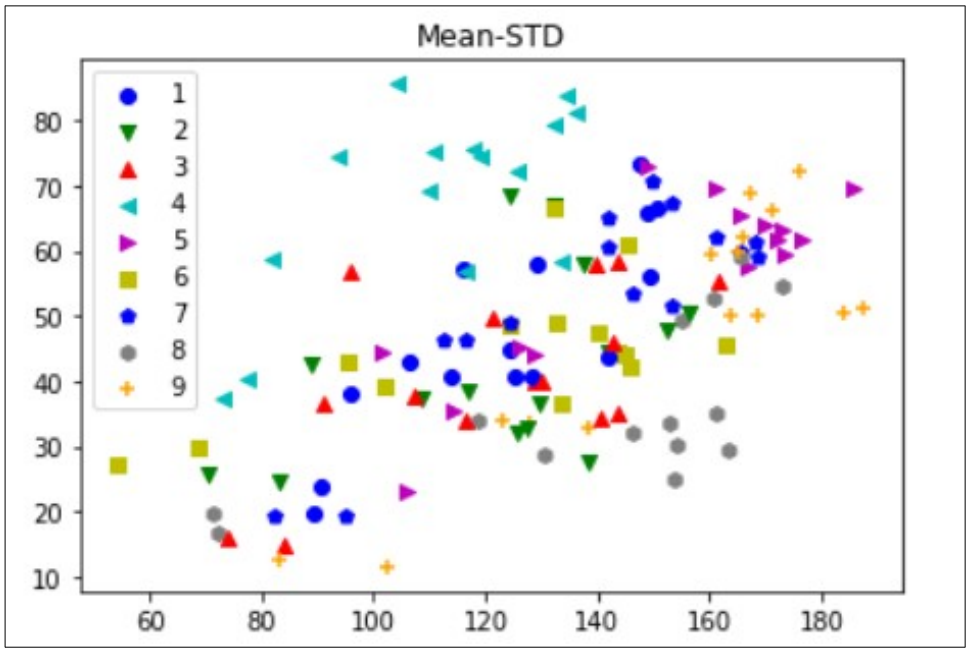


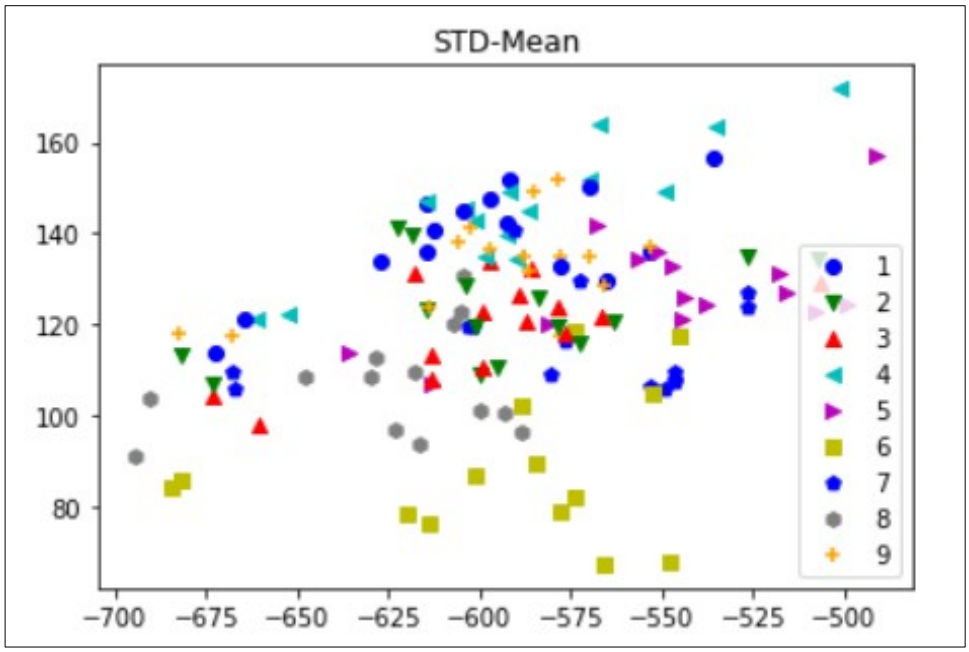
Παρατηρώντας τις παραπάνω γραφικές ανεξαρτήτου ψηφίου ή ομιλητή είναι εμφανές ότι τα χαρακτηριστικά των MFSC έχουν μεγαλύτερη συσχέτιση μεταξύ τους , αυτό μπορούμε να το συμπεράνουμε από τις μεγαλύτερες “ομαλές” περιοχές στις παραπάνω γραφικές που δημιουργούν περιοχές-συστοιχίες πιξελ με ίδιες τιμές ιδίως σε γειτονικά πίξελ και στα “υψηλότερα” χαρακτηριστικά η συσχέτιση είναι ιδιαίτερα μεγάλη. Από την άλλη στους πίνακες των MFCC χαρακτηριστικών βλέπουμε ότι οι τιμές είναι πολύ λιγότερο συσχετισμένες μεταξύ τους (πιο πολλά σκουρόχρωμα πίξελ) και ιδίως μεταξύ γειτονικών χαρακτηριστικών έχουμε αρκετά διαφορετικές τιμές. Τα παραπάνω μας οδηγούν στα να έχουμε δεδομένα που προσδίδουν μεγαλύτερη διακριτική ικανότητα (είναι λιγότερα συσχετισμένα μεταξύ τους) σε σχέση με MFSCs. Το οποίο τα καθιστά καλύτερα ως χαρακτηριστικά καθώς μας “δίνουν” περισσότερη πληροφορία το οποίο είναι επιθυμητό.

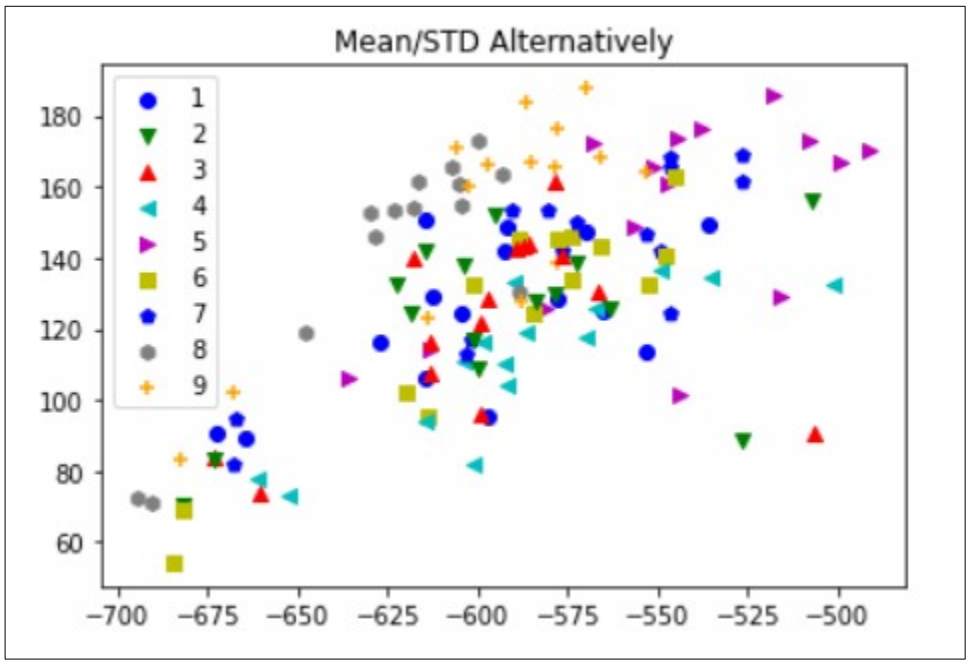
Βήμα 5:

Αφού ενώσουμε τα mfccs-deltas-deltas-deltas μεγέθους #_of_windows x (13+13+13) -> #_of_windows x 39 ,υπολογίζουμε την μέση τιμή και την τυπική απόκλιση τους μεγέθους 1 x 39 το κάθε ένα. Αυθαίρετα, για να κατασκευάσουμε το διάνυσμα, δοκιμάσαμε 3 περιπτώσεις όπου το διάνυσμα που κατασκευάζουμε αποτελείται από τα 39 δείγματα μέσης τιμής και στην συνέχεια τα 39 δείγματα τυπικής απόκλισης, την “ανάποδη” περίπτωση καθώς και την περίπτωση να έχουμε εναλλάξ τα δείγματα μέσης τιμής και τυπικής απόκλισης (όλα τα παραπάνω γίνονται για κάθε εκφώνηση).

Στην συνέχεια παίρνουμε τις 2 πρώτες διαστάσεις του διανύσματος σε κάθε περίπτωση και δημιουργούμε τα scatter plot τους. Τα αποτελέσματα φαίνονται παρακάτω:


Mean-STD
1
2
3
4
5
6
7
8
9
80
70
60
50
40
30
20
10
60
80
100
120
140
160
180


STD-Mean
160
140
120
100
80
−700
−675
−650
−625
−600
−575
−550
−525
−500
1
2
3
4
5
6
7
8
9



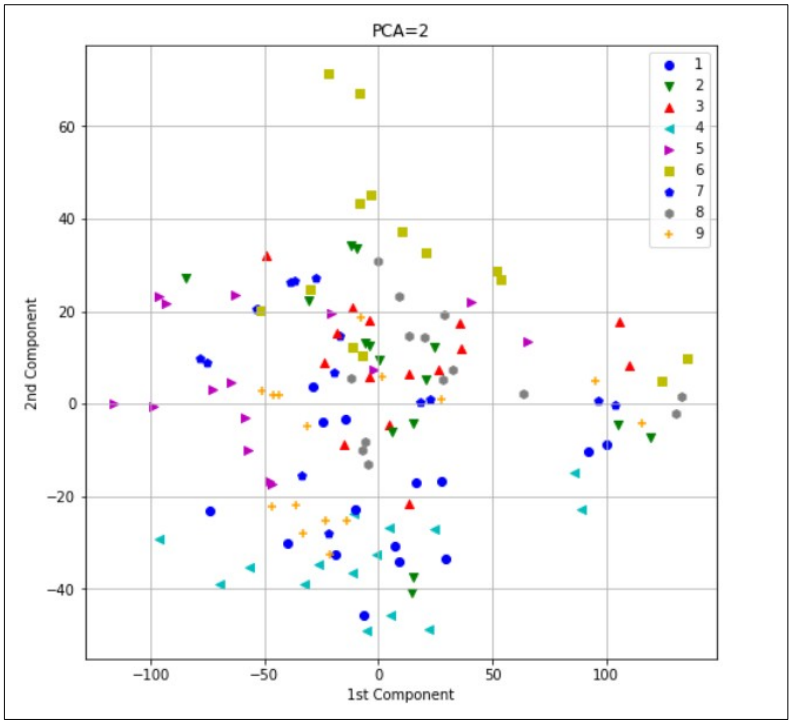
Παρατηρούμε πως σε όλες περιπτώσεις υπάρχει εξαιρετικά μεγάλη ανάμιξη των δειγμάτων με βάση τα features που έχουμε επιλέξει για να τα περιγράψουμε. Συνεπώς δεν υπάρχει καμία περιοχή/συστάδα που θα μπορούσαμε να ορίσουμε η οποία να διαχωρίζει “καλά” τα ψηφία μεταξύ τους.

Βήμα 6:

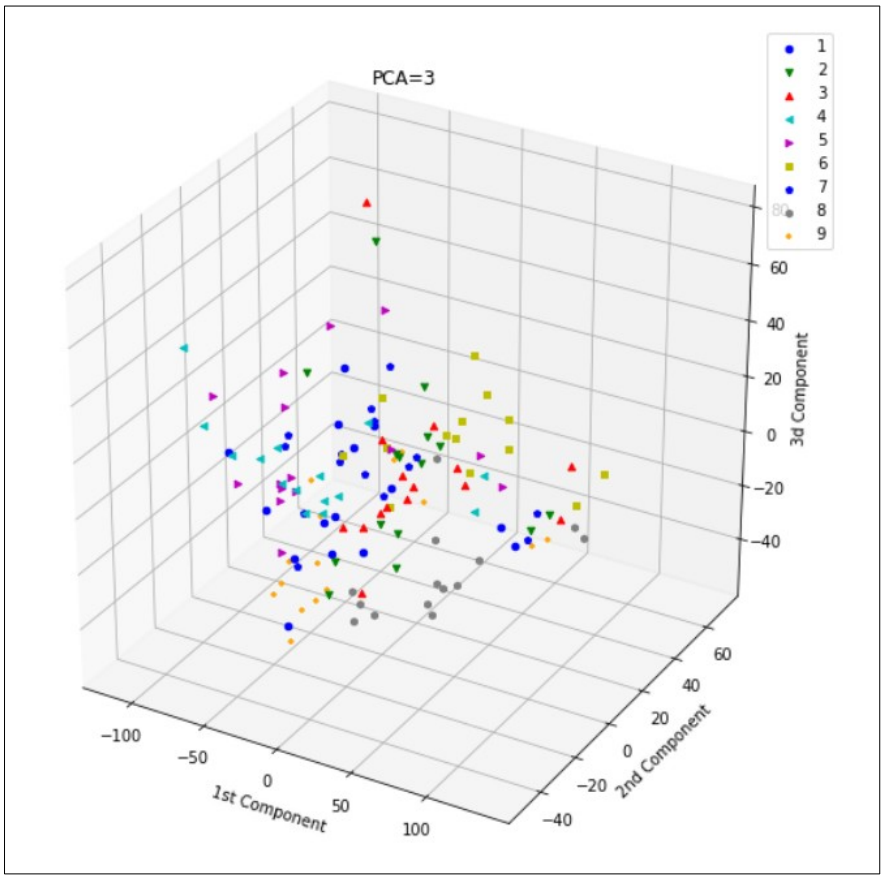
Από τα παραπάνω ,επιλέγουμε να κρατήσουμε την περίπτωση του διανύσματος χαρακτηριστικών που αποτελείται από τις μέσες τιμές πρώτα και στην συνέχεια τις τιμές τυπικής απόκλισης. Προχωράμε σε μείωση των διαστάσεων ώστε αυτές να είναι ίσες 2 μέσα από Principal Component Analysis (PCA).

Με λίγα λόγια η διαδικασία του PCA είναι η εξής: Αρχικά υπολογίζουμε τον πίνακα συσχέτισης (covariance matrix) των μεταβλητών που έχουμε στα δεδομένα. Από αυτόν τον πίνακα βρίσκουμε τις γραμμικώς συσχετισμένες μεταβλητές και βρίσκοντας τα ιδιοδιανύσματα του πίνακα μπορούμε να μετατρέψουμε τον πίνακα με έναν ορθογώνιο μετασχηματισμό και να βρούμε την βάση του νέου πίνακα. Αυτή η βάση του χώρου αποτελεί ένα νέο σύνολο μεταβλητών που είναι γραμμικά ασυσχέτιστες και ονομάζονται κύριες συνιστώσες.

Η γραφική παράσταση που λαμβάνουμε είναι η εξής:



Όμοια με πριν ακόμα και με τις νέες κύριες συνιστώσες βλέπουμε ότι συνεχίζουμε να έχουμε εξαιρετικά μεγάλη επικάλυψη μεταξύ δειγμάτων ψηφίων και συνεπώς αδυναμία να οριστούν περιοχές απόφασης.
Επαναλαμβάνουμε την διαδικασία ανάλυσης σε κύριες συνιστώσες και αυτή την φορά διατηρούμε τις 3 πρώτες.
Όπου παίρνουμε την παρακάτω τρισδιάστατη γραφική παράσταση:



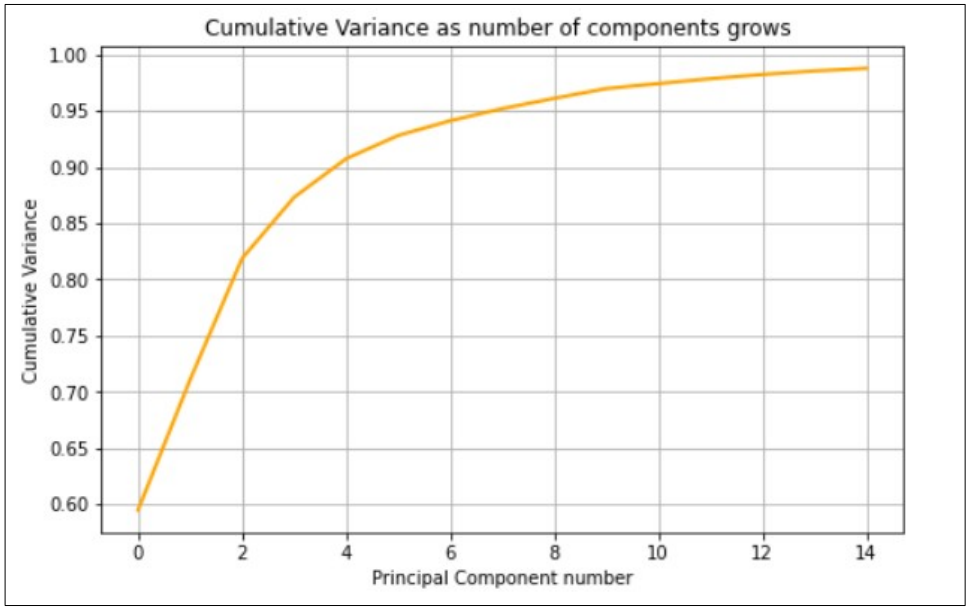
Δυστυχώς ακόμα και με αύξηση σε 3 διαστάσεις και την παρουσίαση τους στο χώρο από αυτά που παρατηρούμε δεν υπάρχει κάποιος καλός διαχωρισμός των δεδομένων σε περιοχές για την διάκριση των διαφορετικών ψηφίων.
Για τις παραπάνω περιπτώσεις βρίσκουμε ότι οι τιμές διασποράς που διατηρούν οι συνιστώσες είναι:

```
PCA with 2 components gives 0.7107913434835774% cumulative Variance
PCA with 3 components gives 0.8187258082767072% cumulative Variance
```

Η διασπορά αυτή εκφράζει το ποσοστό της διασποράς των αρχικών δεδομένων που εκφράζεται με την χρήση 2 και 3 κύριων συνιστωσών. Στην συνέχεια υπολογίζουμε πόσες συνιστώσες θα χρειαζόμασταν για να έχουμε 95% διασπορά , όπου και βρίσκουμε:

```
For 95% variance we need 8 components
```

Επιπλέον κατασκευάζουμε μια γραφική παράσταση για τις τιμές του variance όπως αυτές εξελίσσονται όπως αυξάνουμε τον αριθμό των κύριων συνιστωσών που χρησιμοποιούμε για την μείωση διάστασης.



Από τα ποσοστά καθώς και από τα αποτελέσματα που λάβαμε στις 2 scatter γραφικές καταλήγουμε πως η μείωση σε καμία από τις 2 περιπτώσεις δεν είναι πετυχημένη.

Βήμα 7:

Στην συνέχεια χωρίζουμε τα δεδομένα μας σε Train και Test set , με χρήση της συνάρτησης split_data που κατασκευάσαμε έτσι ώστε να διατηρηθεί η κατανομή των ψηφίων σε κάθε set (θα μπορούσε να γίνει χρήση της sklearn.train_test_split() με την επιπρόσθετη παράμετρο stratify). Στην συνέχεια τα δεδομένα κανονικοποιούνται έτσι ώστε να έχουμε μηδενική μέση τιμή και διασπορά ίση με 1. Το παραπάνω γίνεται με χρήση της StanardScaler της sklearn.preprocessing όπου και εκπαιδεύουμε (fit) πάνω στο σύνολο

εκπαίδευσης και στην συνέχεια εφαρμόζουμε την κανονικοποίηση στο σύνολο εκπαίδευσης και στο σύνολο ελέγχου. Στην συνέχεια με το σύνολο εκπαίδευσης , εκπαιδεύουμε τους εξής ταξινομητές:

1. Custom Naive Bayes
2. Sklearn Naive Bayes
3. kNN με k=3
4. Linear SVM
5. Multilayer Perceptron

Τα αποτελέσματα των ταξινομητών (στο σύνολο ελέγχου) είναι τα παρακάτω:

```
Custom (Gaussian) Naive Bayes Classifier has accuracy: 0.7222222222222222
```

```
Sklearn Gaussian Naive Bayes Classifier has accuracy: 0.7222222222222222
```

```
KNN (k=3) Classifier has accuracy: 0.5555555555555556
```

```
Linear SVM Classifier has accuracy: 0.8611111111111112
```

```
MLP Classifier has accuracy: 0.8888888888888888
```

Βλέπουμε ότι τα καλύτερα αποτελέσματα τα έχουμε για τον MLP (όπου θέσαμε 20 νευρώνες ανά κρυφό επίπεδο και 20 κρυφά επίπεδα) καθώς και σχεδόν ίδια αποτελέσματα για την linear SVM ταξινομητή. Τα χειρότερα αποτελέσματα βρήκαμε με το KNN με 3 γείτονες ωστόσο για 9 γείτονες τα αποτελέσματα μας είναι παρόμοια με του GNB ταξινομητή. Ωστόσο σημαντικό να αναφερθεί για τα παραπάνω πως δεν έγινε κάποια προσπάθεια βελτιστοποίησης εφόσον αυτό δεν αποτελεί στόχος του βήματος αυτού. Επιπλέον παρατηρήσαμε ότι διαφορετικά split των δεδομένων (κρατώντας πάντα την κατανομή των ψηφίων ίδια) οδηγούμαστε σε σημαντικά διαφορετικά αποτελέσματα, το οποίο είναι αναμενόμενο έως ένα βαθμό λόγω της τυχαιότητας του διαχωρισμού. Ωστόσο η ιδιαίτερα μεγάλες αλλαγές θεωρούμε πως οφείλονται και στο γεγονός ότι το σύνολο δεδομένων μας είναι αρκετά μικρό. Συνεπώς πιο εύκολα μπορεί να γίνει ένας “κακός” διαχωρισμός.

Βήμα 8:

Αρχικά για το ερώτημα αυτό δημιουργούμε 500 ακολουθίες ημίτονου και συνημιτόνου με συχνότητα *f=40Hz* με χρήση της συνάρτησης sinData(). Η συνάρτηση αυτή, αρχικά, κατασκευάζει 500 διαφορετικές μετατοπίσεις φάσης και στην συνέχεια για κάθε μία μετατόπιση φάσης κατασκευάζει τις ακολουθίες ,10 σημείων, ημίτονων και συνημίτονων . Σημειώνεται πως κάθε ζεύγος ημίτονου και συνημιτόνου που κατασκευάζεται έχει την ίδια συχνότητα και την ίδια μετατόπιση φάσης.
Αφού κατασκευάζουμε τις παραπάνω ακολουθίες έχουμε δημιουργήσει 3 κλάσεις για την υλοποίηση των αναδρομικών νευρωνικών δικτύων. Πιο συγκεκριμένα πέρα από την RNN (myRNN κλάση) χρησιμοποιούνται και οι

LSTM (myLSTM κλάση) καθώς και GRU (myGRU κλάση), λίγα λόγια για τα τελευταία παραθέτουμε μετά τα αποτελέσματα.

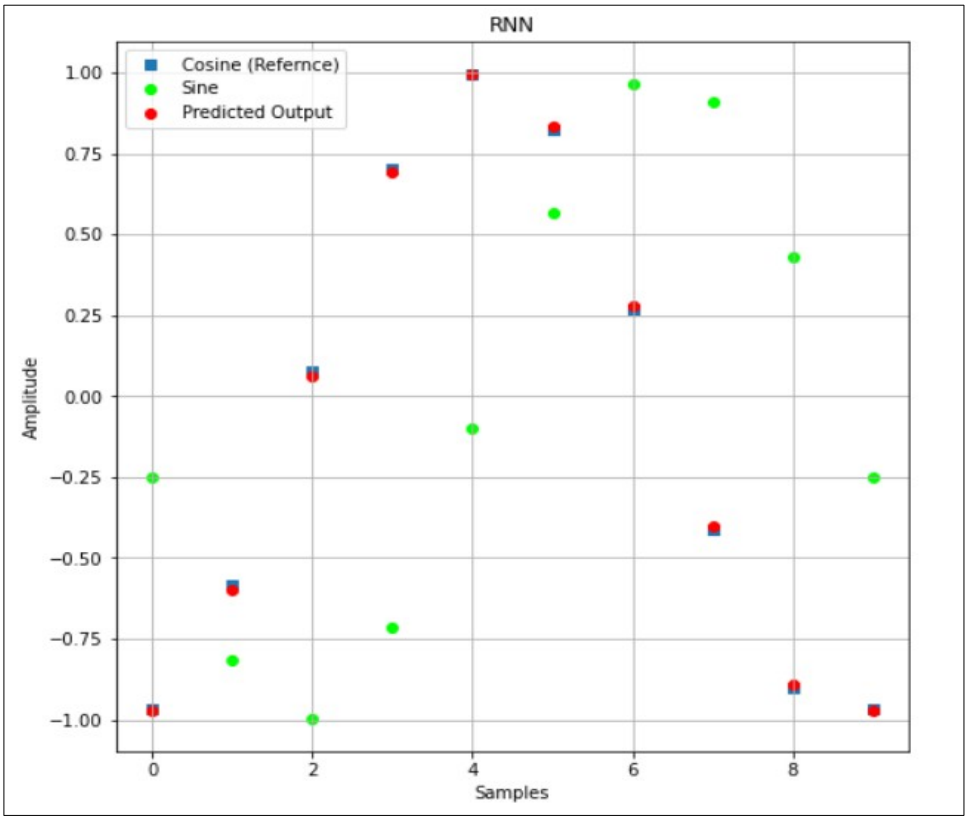
Στο σημείο να αναφέρουμε ότι από τις ακολουθίες που κατασκευάσαμε το 30% γίνεται το test set μας και στην συνέχεια από τα υπόλοιπα κρατάμε 8% για validation set και οι υπόλοιπες αποτελούν το train set μας. Επιπλέον όλα τα δεδομένα στην διαδικασία του fit (εκπαίδευσης) δίνονται σε batch μεγέθους 64 δειγμάτων.

Αρχικά για το RNN με λίγα λόγια αποτελούν γενίκευση των feedforward networks έτσι ώστε να μπορούν να διαχειριστούν ακολουθιακά δεδομένα. Τα RNN μπορούν να “απομνημονεύσουν” τι έχουν δει προηγουμένως στην ακολουθία για να αποφανθούν για επόμενες τιμές.

Προχωράμε στην εκπαίδευση, αρχικά το RNN εκπαιδεύεται για 200 εποχές με learning rate=0.01, αριθμό από κρυφά επίπεδα ίσο με 8 και μέγεθος κρυφών επιπέδων ίσο με 10. Επιπλέον έχει υλοποιηθεί early stopping που αν για 25 εποχές δεν έχουμε κάποια βελτίωση πάνω στο MSE του validation set (που υπολογίζουμε σε κάθε εποχή) τότε σταματάμε την εκπαίδευση. Το τελευταίο συνέβη στην εκπαίδευση μας , η οποία σταμάτησε στις 95 εποχές με mse ίσο με 0.00013556 στο validation set. Προχωράμε στον υπολογισμό του σφάλματος στο test set όπου βρίσκουμε την παρακάτω τιμή:

```
(RNN)Test set MSE loss is:9.830026829149574e-05
```

Η προφανώς είναι εξαιρετικά καλή.
Τέλος κάνουμε γραφική παράσταση για ένα τυχαίο δείγμα από το σύνολο ελέγχου όπου τυπώνουμε το ημίτονο (με πράσινο χρώμα) , το πραγματικό συνημίτονο που αντιστοιχεί στην είσοδο αυτή (με μπλε χρώμα και τετράγωνο σχήμα) και τέλος με κόκκινο η πρόβλεψη μας.



Όπου βλέπουμε ότι υπάρχει σχεδόν απόλυτη ταύτιση των προβλέψεων μας με τις πραγματικές τιμές που αντιστοιχούν στην είσοδο ,κάτι το οποίο ήταν αναμενόμενο εφόσον είχαμε τόσο μικρό σφάλμα.

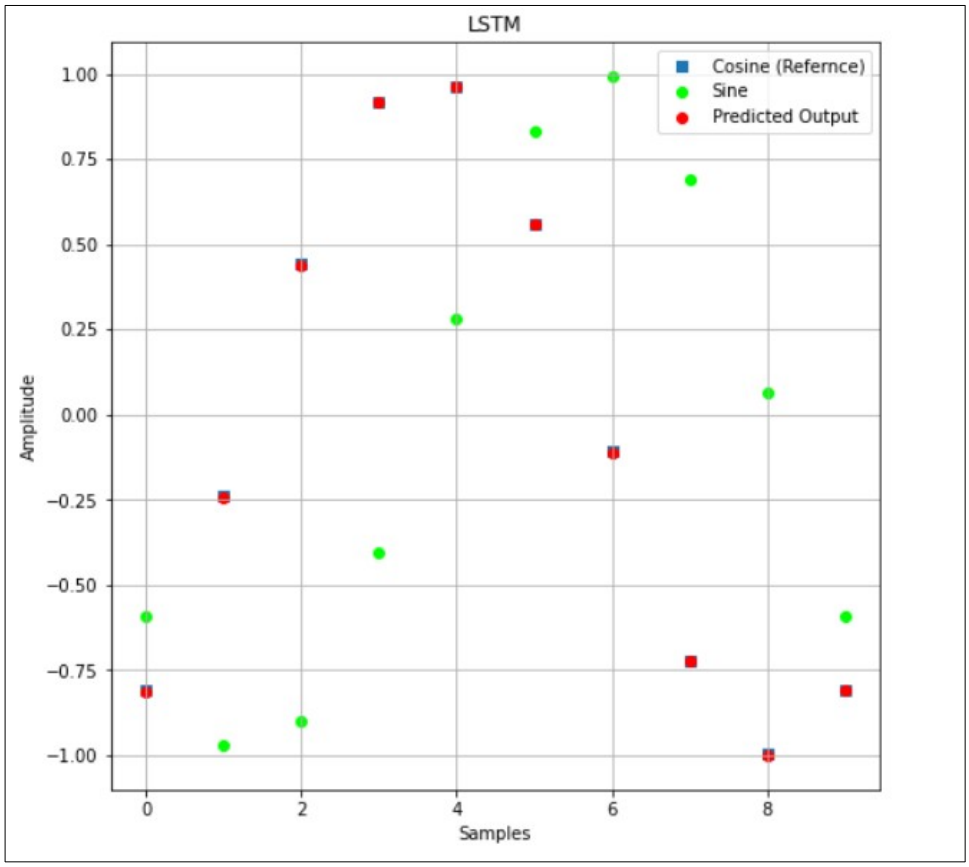
Στην συνέχεια προχωράμε με την ίδια λογική για την υλοποίηση των LSTM και GRU.
Για το LSTM καλούμε για εποχές 150 εποχές, learning rate=0.01,αριθμό κρυφών επιπέδων 4 και μέγεθος κρυφών επιπέδων 10. Η εκπαίδευση σταματά στις 116 εποχές με validation loss:

```
Early Stopping at 116 epochs with validation loss :0.00037766178138554096!
```

και σφάλμα στο σύνολο ελέγχου:

```
(LSTM) Test set MSE loss is:5.108024561195634e-05
```

Τέλος λαμβάνουμε την παρακάτω γραφική για ένα τυχαίο δείγμα του test set:



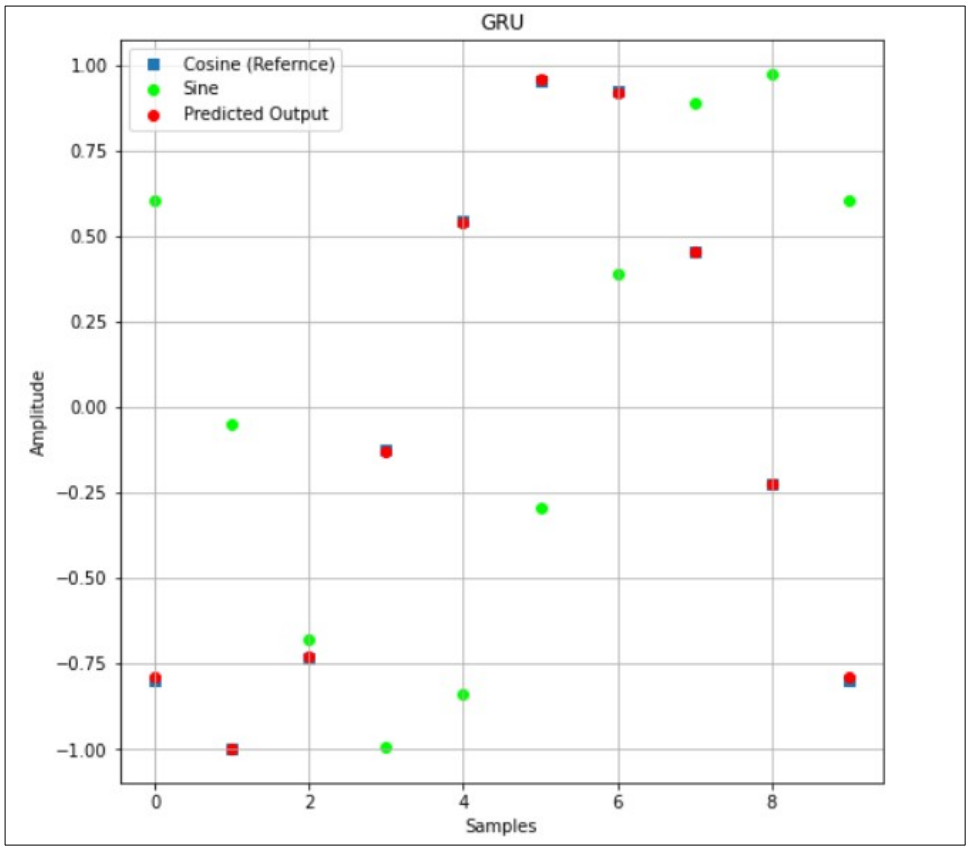
Όμοια με πριν ,όπως περιμέναμε από το πολύ μικρό σφάλμα, έχουμε σχεδόν απόλυτη ταύτιση.

Τέλος για την GRU έχουμε εκπαίδευση με 150 εποχές ,learning rate=0.01 , αριθμό κρυφών επιπέδων ίσο με 5 και μέγεθος κρυφών επιπέδων ίσο με 10 όπου παίρνουμε:

```
Early Stopping at 114 epochs with validation loss :9.90717817330733e-05!
```

```
(GRU) Test set MSE loss is:2.5933202778105624e-05
```

και τελικά η γραφική που παίρνουμε είναι:



Στο σημείο αυτό να αναφέρουμε ότι οι κανονικά τα RNN “υποφέρουν” από short-term memory, καθώς αν μια ακολουθία είναι μεγάλη κατά την διαδικασία του back propagation τα RNN εμφανίζεται το φαινόμενο του Vanishing gradient. Το οποίο οδηγεί με την σειρά του να μειώνεται η ικανότητα εκπαίδευσης όσο προχωρά η διαδικασία του back propagation.

Στην αντιμετώπιση των παραπάνω χρησιμοποιούνται τα LSTMs και GRU’s ως λύση σε “βραχυπρόθεσμη” μνήμη των RNN. Με λίγα λόγια έχουν εσωτερικούς μηχανισμούς ,που ονομάζονται πύλες η οποίες μπορούν να “κανονικοποιήσουν” την ροή πληροφορίας , να μάθουν δηλαδή ποια δεδομένα σε μια ακολουθία είναι χρήσιμα και ποια όχι και συνεπώς κρατώντας μόνο την “σημαντική” πληροφορία. Με τον παραπάνω τρόπο επιλύεται σε έναν βαθμό το πρόβλημα του Vanishing gradient.
Τα αποτελέσματα μας ωστόσο στο πρόβλημα μας είναι εξαιρετικά καλά σε κάθε περίπτωση συνεπώς η επιπλέον πολυπλοκότητα που εισάγουν τα LSTMs και GRUs δεν είναι ανάλογη της βελτίωσης των επιδόσεων, αναφερόμενοι πάντα στο συγκεκριμένο πρόβλημα και σύνολο δεδομένων.

Κύριο Μέρος

Στο σημείο αυτό της εργαστηριακής άσκησης, το ενδιαφέρον μας θα εστιάσουμε στην αναγνώριση ακουστικών χαρακτήρων με βάση τα χαρακτηριστικά που μελετήσαμε στο τμήμα της προεργασίας. Για την ταξινόμηση των ακουστικών ψηφίων θα ακολουθήσουμε δύο διακριτές προσεγγίσεις. Η πρώτη προσέγγιση βασίζεται σε μαρκοβιανά μοντέλα στα οποία τα χαρακτηριστικά των δεδομένων θεωρούνται πιθανές καταστάσεις στις οποίες μπορούν τα μοντέλα να βρεθούν και μοντελοποιούνται ως μείγματα γκαουσιανών κατανομών. Τα μοντέλα αυτά είναι γνωστά ως GMM-HMM . Η δεύτερη προσέγγιση ,στο ζήτημα της αναγνώρισης των ακουστικών ψηφίων, είναι η αξιοποίηση νευρωνικών δικτύων και συγκεκριμένα των LSTM.

Βήμα 9

Προτού ξεκινήσουμε την κατασκευή των μοντέλων μας, με την βοήθεια της συνάρτησης parser που βρίσκεται στο δοθέν αρχείο κώδικα parser.py, πραγματοποιούμε την ανάγνωση των αρχείων ήχου .wav. Επιπλέον, η συνάρτηση αυτή εκτελεί την διαδικασία εξαγωγής των 6 fetaures με τα οποία περιγράφουμε κάθε εκφώνηση και τελικά επιστρέφει τα δεδομένα μας στην επιθυμητή μορφή χωρισμένα σε Train και Test set.

Στην συνέχεια, προκειμένου να αποκτήσουμε και validation set ,το οποίο θα χρησιμοποιηθεί για το fine tuning των μοντέλων που θα υλοποιηθούν, καλούμε την συνάρτηση train_test_split του πακέτου sklearn.model_selection με ορίσματα το Train dataset , τα labels που αντιστοιχούν σε κάθε δείγμα του συνόλου αυτού, το ποσοστό του μεγέθους του Train dataset που θέλουμε να έχει τα προκύπτον validation set. Επίσης, για να εξασφαλίσουμε πως κατά τον διαχωρισμό των δεδομένων τα δύο προκύπτοντα υποσύνολα διατηρούν ίδιο αριθμό διαφορετικών ψηφίων το καθένα , αναθέτουμε στην παράμετρο stratify της παραπάνω συνάρτησης τα σύνολο των labels κάθε δείγματος του αρχικού συνόλου δεδομένων.

Το αποτέλεσμα της παραπάνω κλήσης της συνάρτησης train_test_split είναι ο διαχωρισμός των δεδομένων που είχαμε αρχικά στο σύνολο εκπαίδευσης σε δύο νέα σύνολο , το νέο σύνολο εκπαίδευσης που περιέχει το 80% των αρχικών δειγμάτων και το validation set που περιέχει το εναπομείναν 20%.

Βήμα 10

Έχοντας εξαγάγει τα κατάλληλα χαρακτηριστικά από κάθε εκφώνηση και έχοντας δημιουργήσει τα σύνολο δεδομένων Train set , Test set , Validation set είμαστε σε θέση να ξεκινήσουμε την διαδικασία αναγνώρισης των ακουστικών ψηφίων. Το πρώτο βήμα που πρέπει να πραγματοποιηθεί είναι ο ορισμός ενός GMM-HMM μοντέλου για κάθε ένα από τα ψηφία από το μηδέν έως και το εννιά.

Στο αρχείο hmm.py περιέχεται η υλοποίηση της κλάσης HMM_GMM, η οποία αρχικοποιεί ένα μοντέλο GMM-HMM με n_states κρυφές καταστάσεις και n_mixtures μείγματα γκαουσιανών για την μοντελοποίηση των features κάθε δείγματος. Η κλάση αυτή, διαθέτει επίσης την μέθοδο fit που δέχεται ως όρισμά της το σύνολο δεδομένων εκπαίδευσης (χωρίς τα labels) και με την

οποία πραγματοποιείται η εκπαίδευση ενός GMM-HMM μοντέλου. Συγκεκριμένα, η μέθοδος αυτή λαμβάνει μια λίστα δισδιάστατων numpy arrays , με κάθε array να είναι τα 6 features για κάθε παράθυρο που λάβαμε από το ηχητικό σήμα, και τα συνενώνει με σκοπό την δημιουργία ενός δισδιάστατου array διαστάσεων *συνολικός αριθμός παραθύρων x* 6. Στην συνέχεια, αρχικοποιεί το μείγμα των γκαουσιανών που θα χρησιμοποιηθεί για την μοντελοποίηση των features ως πιθανές καταστάσεις και παράλληλα αποδίδει τυχαίες τιμές στον πίνακα A, εξασφαλίζοντας πως οι πιθανότητες αυτές μετάβασης είναι μη μηδενικές μόνο όταν από την κατάσταση (i,j) είτε μεταβαίνουμε στην (i,j+1) είτε μένουμε στην ίδια. Επιπλέον, η ίδια μέθοδος προτού ξεκινήσει την διαδικασία εκπαίδευσης του μοντέλου, ορίζει πως η πιθανότητα εκκίνησης είναι μη μηδενική και ίση με μονάδα μονό για την κατάσταση 1 και αντίστοιχα πως η πιθανότητα λήξης είναι μη μηδενική και ίση με την μονάδα μόνο για την τελευταία κατάσταση του μοντέλου. Έχοντας ολοκληρώσει τα παραπάνω βήματα, η μέθοδος fit ορίζει την μεταβλητή data η οποία είναι μια λίστα που περιέχει τα δεδομένα εισόδου αποθηκευμένα αυτή την φορά ως λίστα λιστών (και όχι ως 2D numpy array), αρχικοποιεί το GMM-HMM μοντέλο και προχωράει στην εκπαίδευσή του. Τέλος, η κλάση GMM_HMM διαθέτει και την μέθοδο viterbi, η οποία δεχόμενη ένα άγνωστο δείγμα εκτελεί τον αλγόριθμο viterbi και επιστρέφει το λογάριθμο της πιθανότητας το δείγμα αυτό να ανήκει στο ψηφίο ,που μοντελοποιεί το συγκεκριμένο μοντέλο, δεδομένου του πιο πιθανού μονοπατιού καταστάσεων.

Στο σημείο αυτό κρίνεται αναγκαίο να αναφερθεί πως στο κώδικα της εργαστηριακής άσκησης, η εκτέλεση των βημάτων πραγματοποιήθηκε σε κοινό cell προκειμένου να πραγματοποιήσουμε αυτοματοποιημένα την διαδικασία αρχικοποίησης και εκπαίδευσης μοντέλων με διαφορετικές τιμές παραμέτρων n_states και n_mixtures .

Βήμα 11

Με την χρήση ενός διπλού for-loop πραγματοποιούμε την αρχικοποίηση και την εκπαίδευση μοντέλων για κάθε ψηφίο μεταξύ του 0 και του 9, για κάθε συνδυασμό των παραμέτρων $n_{states} \in [1,2,3,4]$ και $n_{mixtures} \in [1,2,3,4,5]$ **.** Τα εκπαιδευμένα μοντέλα για κάθε συνδυασμό παραμέτρων συγκεντρώνονται ,αρχικά, σε μία λίστα η οποία έπειτα αποθηκεύεται σε ένα python dictionary του οποίου τα keys είναι της μορφής $(n_{states}, n_{mixtures})$.

Βήμα 12

Μετά την εκτέλεση του παραπάνω βήματος καταλήγουμε 20 δεκάδες μοντέλων (σε κάθε δεκάδα κάθε μοντέλο αντιστοιχεί σε ένα ψηφίο). Προκείμενου να ανιχνεύσουμε το καλύτερο με την καλύτερη απόδοση, προχωράμε στην υλοποίηση των συναρτήσεων *predict* και *score.* Η συνάρτηση *predict* ,για κάποιο συνδυασμό των υπερπαραμέτρων, εφαρμόζει τον αλγόριθμο viterbi για κάθε δείγμα του validation set στα μοντέλα κάθε ψηφίου με σκοπό την κατασκευή μίας λίστας πιθανοφανειών. Από την λίστα αυτή, προφανώς, ως πρόβλεψη του ταξινομητή θεωρείται το ψηφίο εκείνο το μοντέλο του οποίου έδωσε την μεγαλύτερη πιθανοφάνεια. Επιπλέον , σημειώνεται πως οι προβλέψεις του “ταξινομητή” (συνονθύλευμα μοντέλων

για δεδομένα συνδυασμό υπερπαραμέτρων) αποθηκεύονται σε μία λίστα , η οποία και επιστρέφεται μόλις εξεταστούν όλα τα δείγματα του συνόλου.

Στην συνέχεια, η συνάρτηση score δεχόμενη τα ίδια ορίσματα με την predict, καλεί την predict προκειμένου να υπολογίσει τις προβλέψεις του ταξινομητή , ώστε να συγκρίνει αυτές με την λίστα που περιέχει τα πραγματικά labels για τα δείγματα του Validation set και τελικά να υπολογίσει το συνολικό accuracy αυτού ως τον λόγο τον σωστών προβλέψεων προς το συνολικό αριθμό των δειγμάτων.

Αξιοποιώντας,λοιπόν, την συνάρτηση *score* που μόλις ορίσαμε εκτελούμε ξανά ένα διπλό for-loop πάνω σε κάθε συνδυασμό πιθανών τιμών των υπερπαραμέτρων μας προκειμένου να υπολογίσουμε το accuracy score του εκάστοτε ταξινομητή.

Παρακάτω παρατίθενται τα ποσοστά ακριβείας για κάθε ένα από τους συνδυασμούς των υπερπαραμέτρων που εξετάσαμε.

```
Accuracy of HMM-GMM with 1 hidden state and 1 Gaussian  : 0.8111
Accuracy of HMM-GMM with 1 hidden state and 2 Gaussians : 0.8648
Accuracy of HMM-GMM with 1 hidden state and 3 Gaussians : 0.9037
Accuracy of HMM-GMM with 1 hidden state and 4 Gaussians : 0.9111
Accuracy of HMM-GMM with 1 hidden state and 5 Gaussians : 0.9574
----------------------------------------------------------------
Accuracy of HMM-GMM with 2 hidden states and 1 Gaussian  : 0.8519
Accuracy of HMM-GMM with 2 hidden states and 2 Gaussians : 0.9185
Accuracy of HMM-GMM with 2 hidden states and 3 Gaussians : 0.8963
Accuracy of HMM-GMM with 2 hidden states and 4 Gaussians : 0.9519
Accuracy of HMM-GMM with 2 hidden states and 5 Gaussians : 0.9704
----------------------------------------------------------------
Accuracy of HMM-GMM with 3 hidden states and 1 Gaussian  : 0.8963
Accuracy of HMM-GMM with 3 hidden states and 2 Gaussians : 0.9259
Accuracy of HMM-GMM with 3 hidden states and 3 Gaussians : 0.9648
Accuracy of HMM-GMM with 3 hidden states and 4 Gaussians : 0.9852
Accuracy of HMM-GMM with 3 hidden states and 5 Gaussians : 0.9833
----------------------------------------------------------------
Accuracy of HMM-GMM with 4 hidden states and 1 Gaussian  : 0.9370
Accuracy of HMM-GMM with 4 hidden states and 2 Gaussians : 0.9259
Accuracy of HMM-GMM with 4 hidden states and 3 Gaussians : 0.9778
Accuracy of HMM-GMM with 4 hidden states and 4 Gaussians : 0.9833
Accuracy of HMM-GMM with 4 hidden states and 5 Gaussians : 0.9815
----------------------------------------------------------------
```

Εξετάζοντας τα παραπάνω αποτελέσματα, παρατηρούμε πως η υλοποίηση μοντέλων τριών κρυφών καταστάσεων στην αλυσίδα και 4 γκαουσιανών για την μοντελοποίηση των χαρακτηριστικών των δειγμάτων, οδηγεί στην κατασκευή του ταξινομητή εκείνου που πετυχαίνει το υψηλότερο accuracy score (0.9852) στο validation set.

Έχοντας,λοιπόν, εντοπίσει τον βέλτιστο συνδυασμό υπερπαραμέτρων, προχωράμε στην κατασκευή του αντίστοιχου ταξινομητή προκειμένου να υπολογίσουμε την ακρίβειά του στο test_set. Τα αποτελέσματα φαίνονται παρακάτω.

```
The best combination of parametres is : 3 hidden state(s) , 4 Gaussian(s)

The accuracy of the best classifier on the test set is :  0.97
```

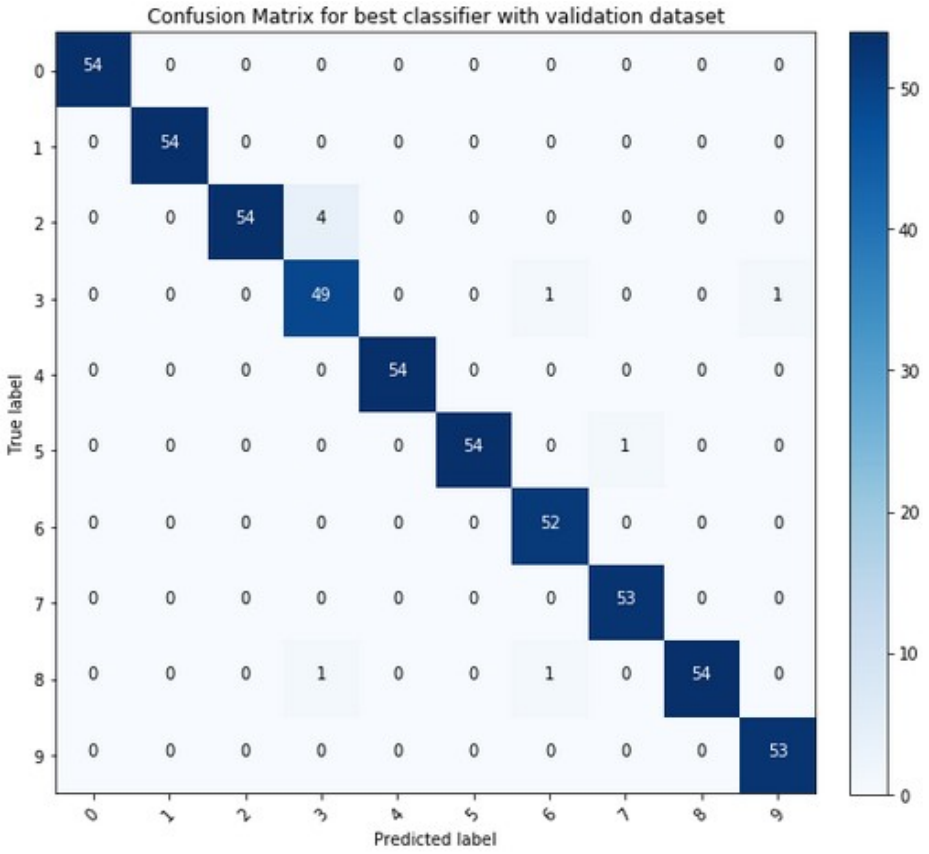
Πριν προχωρήσουμε στην εκτέλεση επόμενων βημάτων της εργαστηριακής αυτής άσκησης, κρίνεται αναγκαίο να αναφερθούμε στην σημαντικότητα της ύπαρξης του validation set για την διεξαγωγή της παραπάνω διαδικασίας.

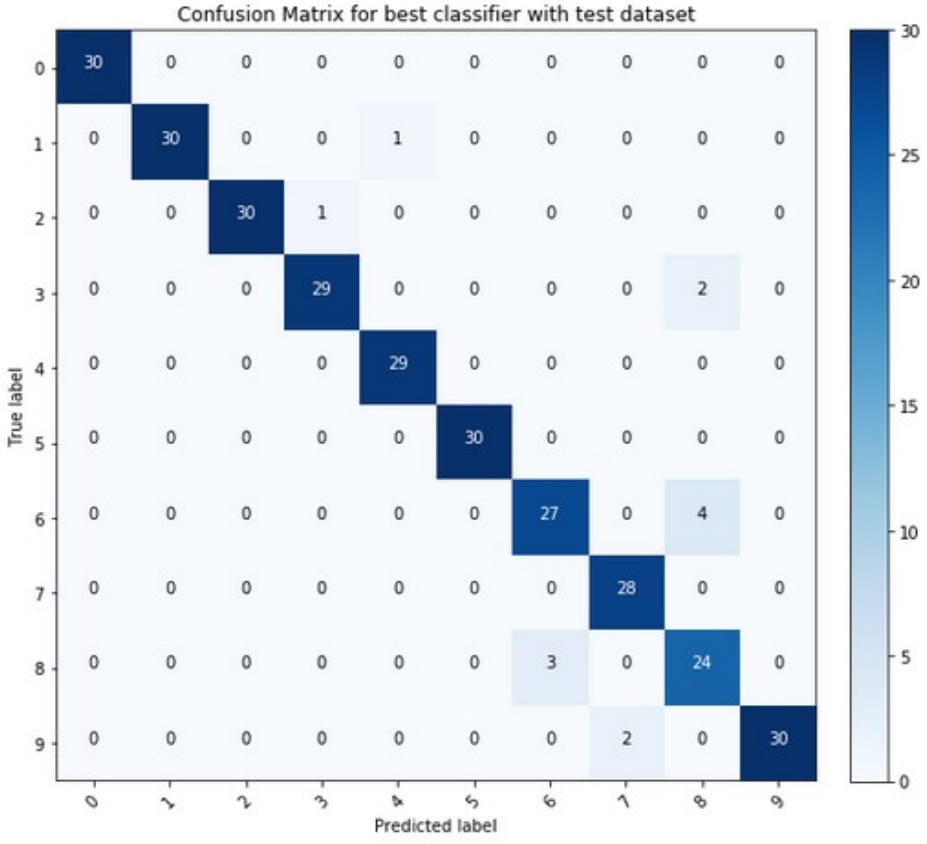
Είναι γνωστό πως για την εύρεση του βέλτιστου συνδυασμού των υπερπαραμέτρων οποιουδήποτε μοντέλου (διαδικασία γνωστή και ως fine tuning), χρειάζεται να εκτελεστούν πολλαπλές εκπαιδεύσεις και αξιολογήσεις του μοντέλου μέχρι τελικά να βρεθεί το μοντέλο με την βέλτιστη απόδοση ως προς την μετρική που μελετάμε. Ωστόσο, καθ' όλη αυτή την επαναληπτική διαδικασία δοκιμών και αξιολογήσεων, θέλουμε να διατηρήσουμε το test set πραγματικά άγνωστο στο μοντέλο μας , προκειμένου να μην εισάγουμε καμίας μορφής bias στην επιλογή των τελικών υπερπαραμέτρων του μοντέλου μας. Για να ικανοποιήσουμε ,λοιπόν, την απαίτηση του πραγματικά άγνωστου test set πραγματοποιούμε τις διαδοχικές αξιολογήσεις του μοντέλου,όπως προκύπτει για κάθε συνδυασμό τιμών των υπερπαραμέτρων του, πάνω στα δεδομένα του Validation set. Σε περίπτωση που πραγματοποιήσουμε τις διαδοχικές αυτές αξιολογήσεις του μοντέλου μας, ελοχεύει ο κίνδυνος το τελικό μοντέλο να αποδίδει καλά πάνω στο δεδομένα test set , αλλά να αδυνατεί να διατηρεί το ίδιο καλές επιδόσεις σε άλλα (πραγματικά άγνωστα αυτή την φορά) δεδομένα.

Βήμα 13

Ένα ιδιαίτερα χρήσιμο εργαλείο στο πεδίο της μηχανικής μάθησης είναι ο πίνακας σύγχυσης (Confusion Matrix), ο οποίος είναι ένας τετραγωνικός δισδιάστατος πίνακας μεγέθους ίσο με το πλήθος των κλάσεων που συμμετέχουν στο πρόβλημα της αναγνώρισης. Ο συγκεκριμένος πίνακας, στην κάθε του διάσταση περιέχει διατεταγμένες σε αύξουσα σειρά τις κλάσεις που εμπλέκονται στο πρόβλημα , με την οριζόντια διάστασή του να αντιστοιχεί στις προβλέψεις του μοντέλου και την κατακόρυφη διάσταση στις πραγματικές ετικέτες των δειγμάτων. Επομένως, ο πίνακας αυτός στο κελί (i,j) περιέχει το πλήθος των δειγμάτων που ανήκουν στην κατηγορία i , αλλά το μοντέλα τα κατέταξε στην κατηγορία j. Στην περίπτωσή μας, ο πίνακας σύγχυσης έχει διαστάσεις 10x10 και υπολογίζεται σε κάθε περίπτωση με την βοήθεια της συνάρτησης confusion_matrix του πακέτου sklear.metrics.

Για την οπτικοποίηση του εκάστοτε πίνακα σύγχυσης, αξιοποιήσαμε ως βάση τον κώδικα που παρέχεται στο αρχείο plot_confusion_matrix.py. Αφού εμπλουτίσαμε τον κώδικα του παραπάνω αρχείου, εκτελέσαμε την συνάρτηση plot_confusion_matrix για τον πίνακα σύγχυσης των προβλέψεων του μοντέλου τόσο στο validation set, όσο και στο test_set. Τα αποτελέσματα της κλήσης της συνάρτησης αυτής παρουσιάζονται στην συνέχεια.


Confusion Matrix for best classifier with validation dataset
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
0 54 0 0 0 0 0 0 0 0 0
1 0 54 0 0 0 0 0 0 0 0
2 0 0 54 4 0 0 0 0 0 0
3 0 0 0 49 0 0 1 0 0 1
4 0 0 0 0 54 0 0 0 0 0
5 0 0 0 0 0 54 0 1 0 0
6 0 0 0 0 0 0 52 0 0 0
7 0 0 0 0 0 0 0 53 0 0
8 0 0 0 1 0 0 1 0 54 0
9 0 0 0 0 0 0 0 0 0 53
50 40 30 20 10 0


Confusion Matrix for best classifier with test dataset
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
0 30 0 0 0 0 0 0 0 0 0
1 0 30 0 0 1 0 0 0 0 0
2 0 0 30 1 0 0 0 0 0 0
3 0 0 0 29 0 0 0 0 2 0
4 0 0 0 0 29 0 0 0 0 0
5 0 0 0 0 0 30 0 0 0 0
6 0 0 0 0 0 0 27 0 4 0
7 0 0 0 0 0 0 0 28 0 0
8 0 0 0 0 0 0 3 0 24 0
9 0 0 0 0 0 0 0 2 0 30
30 25 20 15 10 5 0

Βήμα 14

Στο σημείο αυτό της εργαστηριακής άσκησης, προχωράμε στην υλοποίηση της δεύτερης προσέγγισης στην προσπάθεια αναγνώρισης ακουστικών ψηφίων. Συγκεκριμένα, θα προχωρήσουμε στην κατασκευή ενός LSTM αναδρομικού νευρωνικού δικτύου. Ο λόγος που επιλέγουμε να χρησιμοποιήσουμε ένα αναδρομικό νευρωνικό δίκτυο είναι για να μπορέσουμε να συμπεριλάβουμε στην συνολική διαδικασία εκπαίδευσης του δικτύου την ,αρκετά σημαντική, πληροφορία της συσχέτισης των χαρακτηριστικών μιας εκφώνησης ψηφίου. Επιπλέον, αξιοποιούμε LSTM αναδρομικό δίκτυο διότι ,όπως αναφέραμε και στο τμήμα της προπαρασκευής, αυτός ο τύπος δικτύων επιτυγχάνει την αντιμετώπιση του φαινομένου του vanishing gradient κατά την εκτέλεση του back propagation σε κάθε εποχή.

1) Στο αρχείο lstm.py υλοποιούμε δύο κλάσεις. Η πρώτη κλάση ονομάζεται FrameLevel και κατά την αρχικοποίηση του ενός αντικειμένου αυτής της κλάσης υπολογίζονται και αποθηκεύονται τα μήκη (σε πλήθος παραθύρων) καθενός από τα δεδομένα εισόδου. Επιπλέον, πραγματοποιείται και η διαδικασία του zero_padding στα δείγματα που απαιτείται ,ώστε τελικά όλα τα δείγματα του συνόλου δεδομένων που δόθηκε ως είσοδος να έχουν το ίδιο πλήθος παραθύρων. Το zero_padding στα δείγματα, πραγματοποιείται μέσω της συνάρτησης zero_pad_and_stack η οποία βρίσκεται το μέγιστο πλήθος παραθύρων μεταξύ όλων των δειγμάτων και στην συνέχεια σε όσα δείγματα έχουν λιγότερα παράθυρα από το μέγιστο ενισχύονται με κατάλληλο αριθμό μηδενικών numpy arrays διαστάσεων (1,6).

Η δεύτερη κλάση ,BasicLSTM,που ορίζεται στο αρχείο lstm.py, είναι η κλάση ενός LSTM. Με την συγκεκριμένη κλάση, μπορούμε να αρχικοποιήσουμε ένα LSTM το οποίο δέχεται στην είσοδό του δεδομένα με input_dim features , διαθέτει num_layers κρυφά layers τα οποία το καθένα διαθέτει rnn_size νευρώνες. Επίσης έχουμε την δυνατότητα να ορίσουμε αν το LSTM θα είναι bidirectional μέσω την ομώνυμης boolean παραμέτρου, καθώς και την τιμή του dropout. Επιπλέον, η κλάση αυτή διαθέτει και την μέθοδο forward μέσω της οποίας ορίζεται η ροή της πληροφορίας μέσω στο δίκτυο.

2) Για την αρχικοποίηση του LSTM δικτύου μας, αρκεί να δημιουργήσουμε ένα instance της κλάσης BasicLSTM. Επιλέγουμε σε πρώτη φάση τυχαία τις εξής τιμές για τις υπερπαραμέτρους του δικτύου :

- rnn_size = 20
- num_layers = 1

ενώ το πλήθος features των δεδομένων εισόδου τίθεται να είναι ίσο με την τελευταία διάσταση του πρώτου δείγματος του train set, δηλαδή ίσο με 6, και το μέγεθος της εξόδου το LSTM ορίζεται ίσο με 10 ,όσες δηλαδή και οι πιθανές κλάσεις στις οποίες μπορεί να ανήκει ένα δείγμα.

3) Στο σημείο αυτό, ορίζουμε την συνάρτηση fit η οποία δέχεται ως ορίσματά της κάποιο μοντέλο νευρωνικού δικτύου, το πλήθος των εποχών , το ρυθμό εκμάθησης του optimizer, έναν loader για το train set , έναν loader για το validation set, το ποσοστό L2 regularization το οποίο θα πραγματοποιήσει στα δεδομένα, την boolean παράμετρο val που ορίζει αν θα πραγματοποιηθεί

αξιολόγηση του μοντέλου πάνω στο validation set, την boolean παράμετρο earlyStopping που ορίζει αν θα εφαρμοστεί η πρακτική του early stoppng κατα την διαδικασία εκπαίδευσης του δικτύου , την boolean time_b που εισήχθηκε για τις ανάγκες εκτέλεσης του τελευταίου (μπόνους) ερωτήματος, καθώς και τον τύπο της συσκευής στην οποία θα γίνει η εκπαίδευση του δικτύου. Μόλις λοιπόν, κληθεί η συγκεκριμένη συνάρτηση με τα παραπάνω ορίσματα ,αρχικά, ορίζονται η συνάρτηση σφάλματος που θα χρησιμοποιηθεί για τον υπολογισμό του loss καθώς και ο optimizer που θα πραγματοποιεί την ανανέωση των βαρών του δικτύου. Στην περίπτωση μας, επιλέξαμε η συνάρτηση σφάλματος να είναι η CrossEntropy και ο optimizer μας ο Adam. Παράλληλα ορίζονται και ορισμένες μεταβλητές που θα χρησιμοποιηθούν σε επόμενο ερώτημα για την υλοποίηση του early stopping. Στην συνέχεια, ξεκινάει το επαναληπτικό πέρασμα όλων των δεδομένων εκπαίδευσης σε batches μεγέθους 64 δειγμάτων προκειμένου να εκτελεστεί η διαδικασία της εκμάθησης του νευρωνικού μας δικτύου.

Μετά την ολοκλήρωση της εκπαίδευσης του νευρωνικού δικτύου, αν η τιμή της παραμέτρου val είναι True , προχωράμε στην αξιολόγηση του μοντέλου πάνω στο validation set. Σε κάθε περίπτωση, η συνάρτηση fit επιστρέφει στο τέλος της, το εκπαιδευμένο μοντέλο μαζί με τα losses που έχουν παραχθεί (είτε train losses , είτε train losses και validation losses) .

Έχοντας ,λοιπόν, ορίσει και την συνάρτηση fit προχωράμε στην εκπαίδευση του νευρωνικού δικτύου που ορίσαμε στο προηγούμενο υποερώτημα θέτοντας την val στην τιμή False ,ώστε να μην γίνεται validation του μοντέλου και να τυπώνεται αποκλειστικά και μόνο το μέσο loss στο train set σε κάθε εποχή.

Παρακάτω φαίνεται το αποτέλεσμα της παραπάνω κλήσης της συνάρτησης fit.

```
[ ]  1 model_trainned ,losses= fit(model,40,0.05,train_loader,valid_loader,DEVICE=DEVICE)

Epoch 1 : Train loss is 1.578521164355709
Epoch 2 : Train loss is 0.9721122484280591
Epoch 3 : Train loss is 0.7572213655930884
Epoch 4 : Train loss is 0.5825523177108725
Epoch 5 : Train loss is 0.4907612427025428
Epoch 6 : Train loss is 0.39549840846072515
Epoch 7 : Train loss is 0.3661516946563943
Epoch 8 : Train loss is 0.23438822917240695
Epoch 9 : Train loss is 0.20547967596351624
Epoch 10 : Train loss is 0.21820402291985078
Epoch 11 : Train loss is 0.24947459962967433
Epoch 12 : Train loss is 0.22448941455571006
Epoch 13 : Train loss is 0.1997708500904175
Epoch 14 : Train loss is 0.1778247881278443
Epoch 15 : Train loss is 0.25946613229726895
Epoch 16 : Train loss is 0.23764505904142644
Epoch 17 : Train loss is 0.16573229374538248
Epoch 18 : Train loss is 0.10841321163261498
Epoch 19 : Train loss is 0.15569658817042017
Epoch 20 : Train loss is 0.1221377502779278
Epoch 21 : Train loss is 0.11648203070024954
Epoch 22 : Train loss is 0.2530456294725508
Epoch 23 : Train loss is 0.22434088105117692
Epoch 24 : Train loss is 0.17846149409896428
Epoch 25 : Train loss is 0.13241973132974305
Epoch 26 : Train loss is 0.15587863448397749
Epoch 27 : Train loss is 0.16835771446447684
Epoch 28 : Train loss is 0.11519582687859874
Epoch 29 : Train loss is 0.06584478043905796
Epoch 30 : Train loss is 0.06751560989107709
Epoch 31 : Train loss is 0.04302957863395299
Epoch 32 : Train loss is 0.06228694071016664
Epoch 33 : Train loss is 0.07383325645566431
Epoch 34 : Train loss is 0.07279267906159306
Epoch 35 : Train loss is 0.07327568837876518
Epoch 36 : Train loss is 0.0755792480344798
Epoch 37 : Train loss is 0.11879661886851817
Epoch 38 : Train loss is 0.08874730916595396
Epoch 39 : Train loss is 0.11236247971106071
Epoch 40 : Train loss is 0.11447056070977948
```

4) Στην συνέχεια εκτελούμε την εκπαίδευση του ίδιου μοντέλου θέτοντας αυτή την φορά την παράμετρο val στην τιμή True,ώστε τώρα να γίνει και αξιολόγηση του μοντέλου στο validation set. Η αξιολόγηση αυτή γίνεται, ουσιαστικά, μέσω της πραγματοποίησης προβλέψεων για τις κλάσεις των δειγμάτων του validation set με το διαμορφωμένο έως την τρέχουσα εποχή μοντέλο. Παρακάτω φαίνονται τα μέσα losses σε κάθε εποχή τόσο για το train όσο και το validation set.

```
[ ]  1 model_trainned ,losses= fit(model,40,0.05,train_loader,valid_loader,val=True,DEVICE=DEVICE)

Epoch 1 : Train loss is 0.11945278250636909
Epoch 1 : Validation Loss is 0.3365616759439712
---------------------------------------------------
Epoch 2 : Train loss is 0.11874366492860865
Epoch 2 : Validation Loss is 0.2945280089381717
---------------------------------------------------
Epoch 3 : Train loss is 0.1138667808755761
Epoch 3 : Validation Loss is 0.2922964466580756
---------------------------------------------------
Epoch 4 : Train loss is 0.10817226516480719
Epoch 4 : Validation Loss is 0.26923056722085353
---------------------------------------------------
Epoch 5 : Train loss is 0.07267642496670408
Epoch 5 : Validation Loss is 0.24904435472020062
---------------------------------------------------
Epoch 6 : Train loss is 0.09607307178668457
Epoch 6 : Validation Loss is 0.39930552135661956
---------------------------------------------------
Epoch 7 : Train loss is 0.14661967588737415
Epoch 7 : Validation Loss is 0.23834890568321046
---------------------------------------------------
Epoch 8 : Train loss is 0.098579009724958
Epoch 8 : Validation Loss is 0.248441145676832
---------------------------------------------------
Epoch 9 : Train loss is 0.0628100739436448
Epoch 9 : Validation Loss is 0.24425830197417076
---------------------------------------------------
Epoch 10 : Train loss is 0.06890939019949141
Epoch 10 : Validation Loss is 0.2224053628440253
---------------------------------------------------
Epoch 11 : Train loss is 0.0600812938581924
Epoch 11 : Validation Loss is 0.28212462206834527
---------------------------------------------------
Epoch 12 : Train loss is 0.053331425647747475
Epoch 12 : Validation Loss is 0.2007904543376949
---------------------------------------------------
Epoch 13 : Train loss is 0.0719145170624916
Epoch 13 : Validation Loss is 0.44826274305546043
---------------------------------------------------
Epoch 14 : Train loss is 0.2026762714201808
Epoch 14 : Validation Loss is 0.449320601656676626
---------------------------------------------------
Epoch 15 : Train loss is 0.22048186587511392
Epoch 15 : Validation Loss is 0.2964881175967851
---------------------------------------------------
Epoch 16 : Train loss is 0.18029889141440378
Epoch 16 : Validation Loss is 0.5961053751510784
---------------------------------------------------
Epoch 17 : Train loss is 0.4814439655110803
Epoch 17 : Validation Loss is 0.9009102704343344
---------------------------------------------------
Epoch 18 : Train loss is 0.7347268740462048
Epoch 18 : Validation Loss is 0.6694788771036152
---------------------------------------------------
Epoch 19 : Train loss is 0.6723095200631979
Epoch 19 : Validation Loss is 0.9725431771023051
---------------------------------------------------
```

Σημειώνεται πως τα παραπάνω είναι μόνο τα losses μέχρι και την 19η, από τις 40 , εποχή.

Προτού προχωρήσουμε στην αξιοποίηση επιπρόσθετων μεθόδων για την βελτίωση των επιδόσεων του μοντέλου μας, αποφασίσαμε να πραγματοποιήσουμε ένα αρκετά περιορισμένα GridSeach με σκοπό την εύρεση των συνδυασμών εκείνων των παραμέτρων rnn_size και num_layers που οδηγούν στα μοντέλα με τις 4 καλύτερες επιδόσεις ως προς την μετρική accuarcy.

Προκειμένου να μπορέσουμε να βρούμε τους 4 προαναφερθέντες συνδυασμούς ορίσαμε την συνάρτηση score, η οποία δεχόμενη ως παραμέτρους της το μοντέλο που έχει προκύψει από τον εκάστοτε συνδυασμό παραμέτρων και κάποιο loader δεδομένων εκτελεί την διαδικασία της πρόβλεψης της κλάσης στην οποία ανήκει το κάθε δείγμα και τελικά υπολογίζει ,κατά τα γνωστά, την μετρική accuracy.

Έχοντας,λοιπόν, ορίσει και την συνάρτηση που υπολογίζει το accuracy ενός μοντέλου, πραγματοποιούμε την εκπαίδευση των μοντέλων για κάθε συνδυασμό των υπερπαραμέτρων $rnn_{size} \in [20,40,60]$ και $num_{layers} \in [1,2,3]$. Τα εκπαιδευμένα μοντέλα , μαζί με τα losses που συνοδεύουν την εκπαίδευση τους αποθηκεύονται σε ένα python dictionary του οποίου τα keys είναι της μορφής $(rnn_{size}, num_{layers})$. Μετά την ολοκλήρωση της εκπαίδευσης των μοντέλων, επαναληπτικά για κάθε πιθανό συνδυασμό υπερπαραμέτρων υπολογίζουμε το accuracy των προκυπτόντων μοντέλων , ώστε τελικά να κρατήσουμε τους 4 καλύτερους συνδυασμούς. Τα μοντέλα αυτά είναι τα εξής :

```
Number 1 best model(accuracy = 0.9888888888888889) : rnn_size = 60 , num_layers = 3
Number 2 best model(accuracy = 0.9833333333333333) : rnn_size = 60 , num_layers = 2
Number 3 best model(accuracy = 0.9833333333333333) : rnn_size = 40 , num_layers = 1
Number 4 best model(accuracy = 0.9814814814814815) : rnn_size = 60 , num_layers = 1
```

5) Στο σημείο αυτό, εισάγουμε στο δίκτυό L2 regularization και dropout και πραγματοποιούμε GridSearch ,κατά τρόπο παρόμοιο με προηγουμένως, για τα 4 καλύτερα μοντέλα που βρήκαμε από το παραπάνω GridSearch. Οι τιμές για τις παραμέτρους που δοκιμάσαμε είναι οι εξής :

- $dropout \in [0,0.1,0.2,0.3]$
- $regularization_{L2} \in [0,0.0001,0.0002]$

6) Από το προηγούμενο GridSearch εντοπίζουμε τους 8 συνδυασμούς των 4 υπερπαραμέτρων αυτή την φορά που οδηγούν στα μοντέλα με τις καλύτερες επιδόσεις ως προς την μετρική accuracy. Τα μοντέλα αυτά είναι τα εξής :

```
Number 1 best model(accuracy = 0.987037037037037) : rnn_size = 60 , num_layers = 1
Number 2 best model(accuracy = 0.987037037037037) : rnn_size = 60 , num_layers = 1
Number 3 best model(accuracy = 0.987037037037037) : rnn_size = 60 , num_layers = 1
Number 4 best model(accuracy = 0.9851851851851852) : rnn_size = 60 , num_layers = 2
Number 5 best model(accuracy = 0.9851851851851852) : rnn_size = 60 , num_layers = 2
Number 6 best model(accuracy = 0.9851851851851852) : rnn_size = 60 , num_layers = 2
Number 7 best model(accuracy = 0.9851851851851852) : rnn_size = 60 , num_layers = 2
Number 8 best model(accuracy = 0.9851851851851852) : rnn_size = 60 , num_layers = 2
```

Παρατηρούμε πως από αυτά τα 3 πρώτα πετυχαίνουν όλα το ίδιο accuracy , ομοίως και για τα υπόλοιπα 5. Για τον λόγο αυτό, επιλέγουμε να κρατήσουμε μόνο το πρώτο μοντέλο και το τέταρτο μοντέλο. Στα μοντέλα που επιλέξαμε, εισάγουμε την δυνατότητα early stopping. Συγκεκριμένα, για την υλοποίηση του early stopping ,όπως αναφέραμε και κατά την εξήγηση της συνάρτησης fit, εισαγάγαμε ορισμένες επιπρόσθετες μεταβλητές που θα χρησιμοποιηθούν για το early stopping. Οι μεταβλητές αυτές είναι η n_epochs_stop η οποία αντιστοιχεί στο πλήθος των εποχών που αν παρέλθουν χωρίς το μέσο loss του μοντέλου στο validation set να μειωθεί η διαδικασία της εκπαίδευσης ολοκληρώνεται και επιστρέφεται το μοντέλο που είχε πετύχει το ελάχιστο loss. Επιπλέον, χρησιμοποιούμε την μεταβλητή min_val_loss για να αποθηκεύσουμε το ελάχιστο μέχρι τώρα loss στο validation set, καθώς και την μεταβλητή epochs_no_improve που μετρά πόσες εποχές δεν υπάρχει μείωση στο μέσο validation loss.

Έχοντας ορίσει τις μεταβλητές αυτές, η διαδικασία του ελέγχου για early stop είναι αρκετά απλή. Σε κάθε εποχή υπολογίζεται το μέσο validation loss το οποίο στην συνέχεια συγκρίνεται με το ελάχιστο μέσο validation loss, αν αυτό είναι μικρότερο τότε η μεταβλητή epochs_no_improve επαναφέρεται στην τιμή μηδέν και σε μια βοηθητική μεταβλητή best_model αποθηκεύουμε ένα αντίγραφο του μοντέλου όπως έχει διαμορφωθεί μέχρι την δεδομένη εποχή. Σε περίπτωση που το μέσο validation loss είναι μεγαλύτερο του ελάχιστου τότε απλά αυξάνουμε κατά 1 την epochs_no_improve. Μετά την ολοκλήρωση του ελέγχου αυτού, εξετάζουμε αν το epochs_no_improve έχει γίνει μεγαλύτερο του n_epochs_stop και αυτό ισχύει και ταυτόχρονα το πλήθος των εποχών που έχουν εκτελεστεί είναι πάνω από κάποιο ορισθέν κάτω όριο, τότε η διαδικασία της εκτέλεσης τερματίζεται και επιστρέφεται το βέλτιστο μοντέλο, μαζί με τα losses στα δύο datasets. Σε διαφορετική περίπτωση, προχωράει η διαδικασία της εκπαίδευσης σε επόμενη εποχή.

Επομένως, στο σημείο αυτό εκτελούμε ξανά εκπαίδευση ,με την δυνατότητα early stopping,των μοντέλων που προκύπτουν από τους 2 συνδυασμούς των 4 παραμέτρων που επιλέξαμε παραπάνω και κατά τα γνωστά υπολογίζουμε τα accuracy αυτών. Παρακάτω φαίνεται ο συνδυασμός παραμέτρων που οδήγησε στο καθολικά βέλτιστο μοντέλο , καθώς και το accuracy αυτού στο validation set

```
Best model(accuracy = 0.987037037037037) : rnn_size = 60 , num_layers = 1, dropout = 0.3, L2_regularization = 0.0001
```

Επιπλέον, καθώς η διαδικασία εύρεσης του βέλτιστου συνδυασμού παραμέτρων μπορεί να θεωρηθεί ιδιαίτερα χρονοβόρα, αποθηκεύουμε σε ένα pickle το βέλτιστο μοντέλο όπως προέκυψε από την παραπάνω αναζήτηση καθώς και τα losses που αντιστοιχούν στην εκπαίδευση και στο validation αυτού.

7) Συνεχίζοντας την προσπάθεια εύρεσης ενός καθολικά βέλτιστου μοντέλου ως προς την μετρική accuracy, ορίζουμε ένα Bidirectional LSTM με τιμές των παραμέτρων να είναι ίσες με αυτές που βρήκαμε στο προηγούμενο υποερώτημα. Η βασική διαφορά μεταξύ ενός LSTM και ενός BiLSTM είναι ο τρόπος με τον οποίο ρέει η πληροφορία στο εσωτερικό του δικτύου. Συγκεκριμένα, σε ένα απλό LSTM η πληροφορία ρέει από την είσοδο προς την έξοδο με τα hidden cell να διατηρούν την πληροφορία και από προηγούμενα δείγματα που έχει το νευρωνικό μας δίκτυο. Στην περίπτωση του BiLSTM, τα δεδομένα εισόδου ρέουν τόσο από την είσοδο προς την έξοδο ,όσο και από την έξοδο προς την είσοδο , με αποτέλεσμα το δίκτυο αυτό να μπορεί να εκτελεί την διαδικασία της εκπαίδευσής κάθε νευρώνα αξιοποιώντας “γνώση” από το παρελθόν αλλά και από το “μέλλον” (επόμενων hidden cells).

Η ιδιαιτερότητα αυτή του BiLSTM ,σε συνδυασμό με το γεγονός πως ένα παράθυρο κάποιου ακουστικού σήματος εξαρτάται τόσο από τα προηγούμενά του όσο και από τα επόμενα του, είναι εκείνη που μας ώθησε στην χρήση αυτού του νευρωνικού δικτύου κατά την προσπάθεια βελτίωσης ακόμα περισσότερο της ακρίβειας του συστήματος αναγνώρισης ακουστικών ψηφίων.

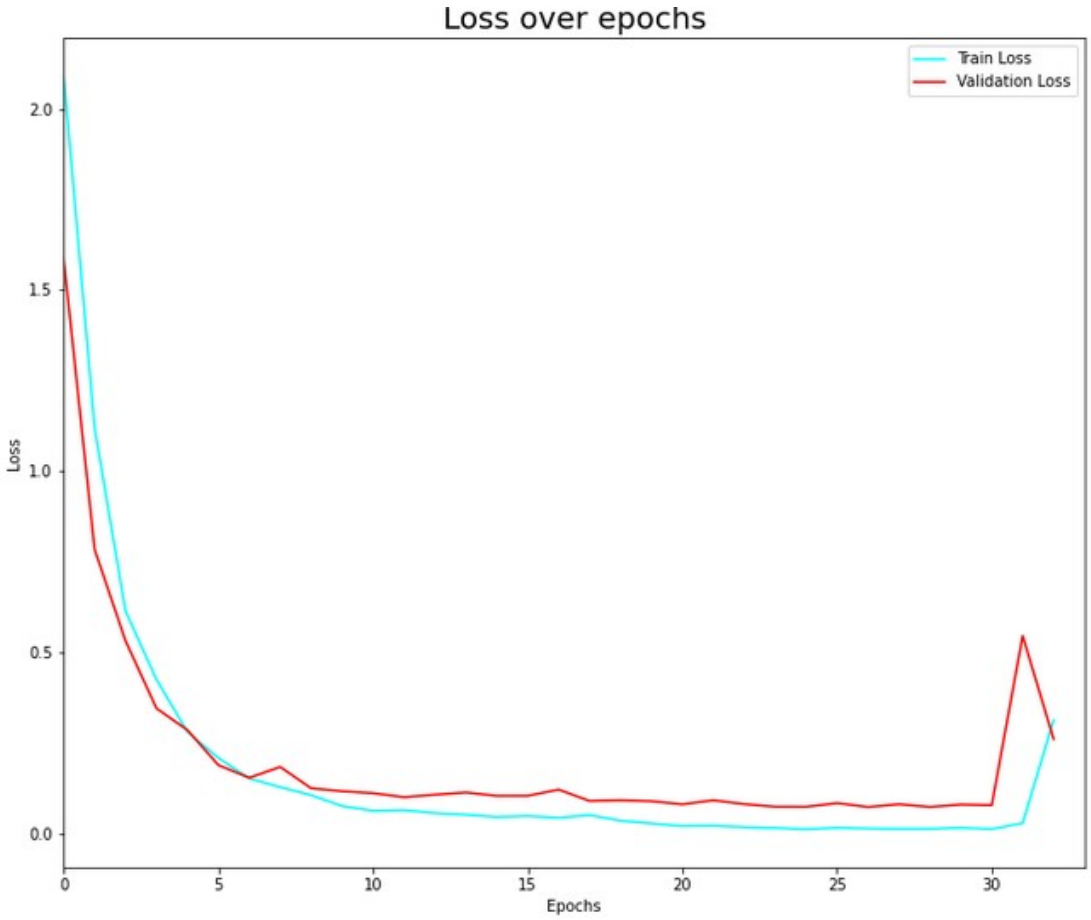
Παρακάτω φαίνεται η ακρίβεια του BiLSTM με τις δεδομένες τιμές των υπερπαραμέτρων του.

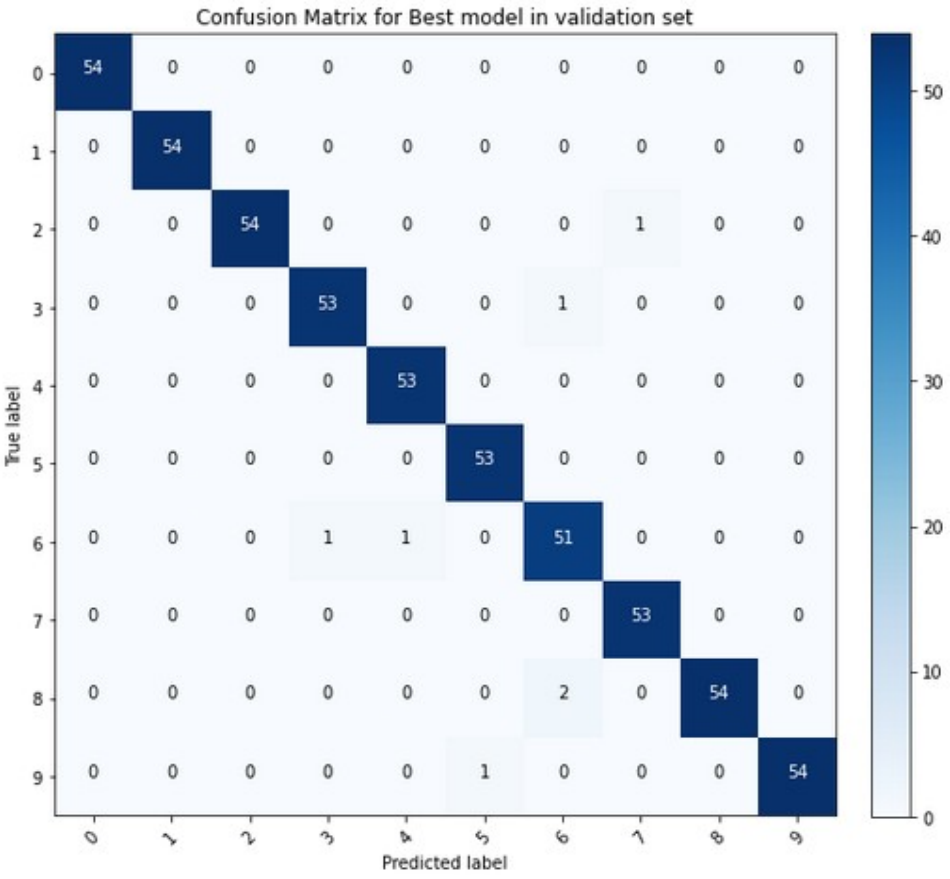
```
# The accuracy achieved by the above BiLSTM.
score_test,preds_test = score(bilstm[0],test_loader)
score_val,preds_val = score(bilstm[0],valid_loader)
print('The accuracy score achieved by the above BiLSTM (in test dataset) is :','{:.2f}'.format(score_test))
```

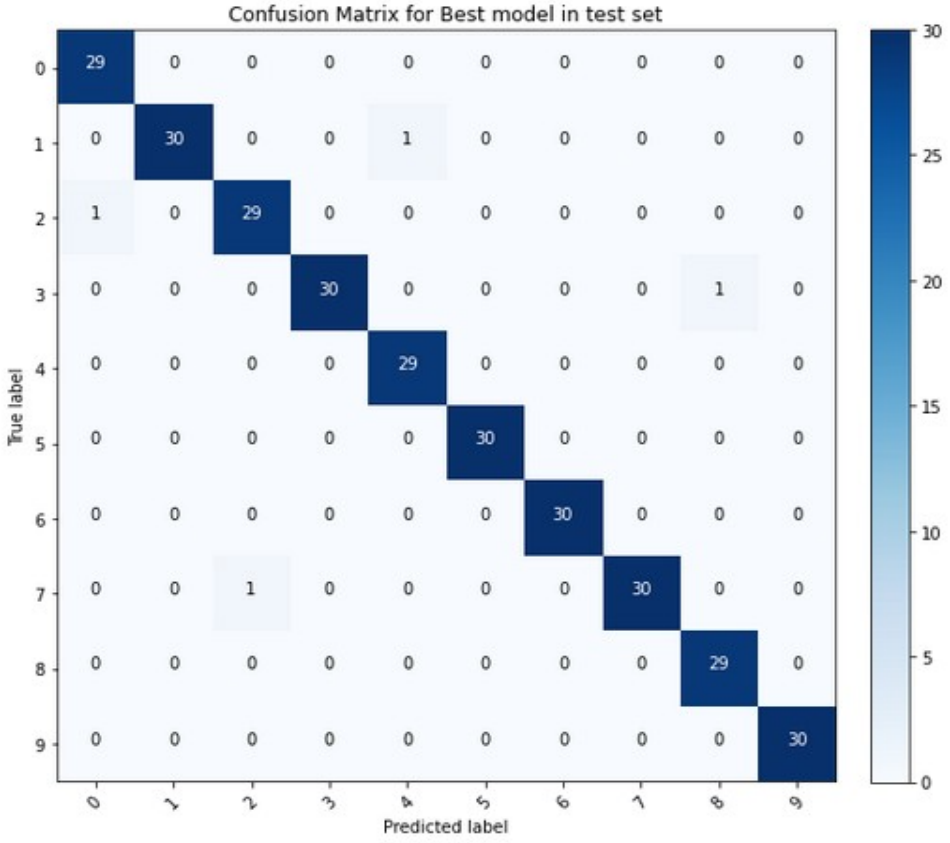
```
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/rnn.py:582: UserWarning: RNN module weights are not pa
  self.dropout, self.training, self.bidirectional, self.batch_first)
The accuracy score achieved by the above BiLSTM (in test dataset) is : 0.99
```

Παρατηρούμε πως το BiLSTM πετυχαίνει accuracy score ίσο με 99% , ενώ το απλό LSTM πετυχαίνει accuracy score ίσο με 98.7%. Επομένως, με ελάχιστη διαφορά ,που μπορεί να οφείλεται και στην στρογγυλοποίηση, το BiLSTM είναι το καθολικά βέλτιστο από τα μοντέλα που εξετάσαμε.

Στην συνέχεια , παραθέτουμε το γράφημα μεταβολής του train και του validation loss συναρτήσει της εποχής, καθώς και τον πίνακα σύγχυσης για το BiLSTM όπως αυτός προκύπτει για το validation set και για το test set αντίστοιχα.




Confusion Matrix for Best model in validation set
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
54 0 0 0 0 0 0 0 0 0
0 54 0 0 0 0 0 0 0 0
0 0 54 0 0 0 0 1 0 0
0 0 0 53 0 0 1 0 0 0
0 0 0 0 53 0 0 0 0 0
0 0 0 0 0 53 0 0 0 0
0 0 0 1 1 0 51 0 0 0
0 0 0 0 0 0 0 53 0 0
0 0 0 0 0 0 2 0 54 0
0 0 0 0 0 1 0 0 0 54
50 40 30 20 10 0


Confusion Matrix for Best model in test set
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
29 0 0 0 0 0 0 0 0 0
0 30 0 0 1 0 0 0 0 0
1 0 29 0 0 0 0 0 0 0
0 0 0 30 0 0 0 0 1 0
0 0 0 0 29 0 0 0 0 0
0 0 0 0 0 30 0 0 0 0
0 0 0 0 0 0 30 0 0 0
0 0 1 0 0 0 0 30 0 0
0 0 0 0 0 0 0 0 29 0
0 0 0 0 0 0 0 0 0 30
30 25 20 15 10 5 0

8) Στο ερώτημα αυτό κάνουμε χρήση του pack_padded_sequence για την εκπαίδευση του μοντέλου. Ο λόγος που εφαρμόζουμε το παραπάνω είναι πως όταν εκπαιδεύουμε ένα RNN (LSTM) δημιουργείται θέμα με το μεταβλητό μήκος ακολουθίας που έχουν τα δεδομένα μας (αριθμός παραθύρων σε κάθε εκφώνηση) προς επίλυση του παραπάνω κάνουμε padding με 0 σε όλα τα δεδομένα ώστε να έχουν μέγεθος ακολουθίας όσο το μεγαλύτερο που συναντάμε στα δεδομένα μας. Το παραπάνω θα οδηγήσει ωστόσο σε πολλαπλάσιες παραπάνω πράξεις που δεν χρειάζεται να γίνουν. Μάλιστα στην περίπτωση μας αν τυπώσουμε την μέση τιμή των μηκών (το άθροισμα των παραθύρων δηλαδή) του κάθε δείγματος ακολουθία (χωρίς βλάβη της γενικότητας για ένα τυχαίο train test val split) παίρνουμε τιμή σχεδόν 30, αλλά αν τυπώσουμε την μέγιστη τιμή, με βάση της οποίας γίνεται το padding σε όλα, παίρνουμε 153. Συνεπώς είναι προφανές ότι γίνονται εξαιρετικά πολλές παραπάνω πράξεις.
Με το pack_padded_sequence μας δίνεται η δυνατότητα να “συμπιέσουμε” την ακολουθία σε ένα tuple 2 λιστών , όπου 1 μια περιέχει όλα τα δεδομένα των ακολουθιών και η άλλη το batch_size της packed ακολουθίας. (Να αναφέρουμε στο σημείο αυτό ότι προφανώς οι συνάρτηση nn.lstm (και λοιπές) δέχονται ως είσοδο packed ακολουθίες).Στόχος του παραπάνω λοιπόν είναι προφανώς να μειώσουμε τις περιττές πράξεις και εν δυνάμει το χρόνο εκπαίδευσης.
Αρχικά για να υλοποιήσουμε το παραπάνω πρέπει να ταξινομήσουμε τις ακολουθίες μας κατα φθίνουσα σειρά μήκους. Υλοποιούμε τα παραπάνω και παίρνουμε τους dataloader train_loader_srt και valid_loader_srt για το train και validation set αντίστοιχα. Στην υλοποίηση της LSTM έχουμε ορίσει μια default παράμετρο pad η οποία όταν παίρνει τιμή True εφαρμόζει την packed_padded_sequence στο batch που του έχουμε δώσει (τα δεδομένα είναι ήδη σε φθίνουσα σειρά μήκους από την υλοποίηση του dataloader). Τα μετασχηματισμένα δεδομένα τα δίνουμε στην lstm συνάρτηση της οποίας το αποτέλεσμα είναι σε packed μορφή. Για να συνεχίσουμε στα υπόλοιπα βήματα κάνουμε pad_packed_sequence το αποτέλεσμα για να είναι ξανά σε μορφή όπως αρχικά. Εφαρμόζοντας τα παραπάνω προχωράμε σε εκτύπωση των χρόνων εκπαίδευσης που καταλήγουμε στην ενδιαφέρουσα παρακάτω παρατήρηση
Ο χρόνος για την διαδικασία forward πράγματι μειώνεται όπως βλέπουμε παρακάτω (παίρνουμε το μέσο όρο για μια εποχή).

Με pack_padded_sequence:
Για batch_size=64:

```
Epoch 1 mean of forward time is :0.006362315380212032
Epoch 2 mean of forward time is :0.00643665862805915 6
Epoch 3 mean of forward time is :0.006382356990467419
Epoch 4 mean of forward time is :0.006522503766146573
Epoch 5 mean of forward time is :0.006336876840302439
```

Για batch_size=512:

```
Epoch 1 mean of forward time is :0.02185356616973877
Epoch 2 mean of forward time is :0.021716415882110596
Epoch 3 mean of forward time is :0.022188246250152588
Epoch 4 mean of forward time is :0.021357476711273193
Epoch 5 mean of forward time is :0.021594882011413574
```

Χωρίς:
Για batch_size=64:

```
Epoch 1 mean of forward time is :0.02891185066916726
Epoch 2 mean of forward time is :0.02936279412471887
Epoch 3 mean of forward time is :0.0301253578879616 47
Epoch 4 mean of forward time is :0.029159054611668442
Epoch 5 mean of forward time is :0.029551925081195 25
```

Για batch_size=512:

```
Epoch 1 mean of forward time is :0.12316316366195679
Epoch 2 mean of forward time is :0.11966574192047119
Epoch 3 mean of forward time is :0.12190765142440796
Epoch 4 mean of forward time is :0.12064361572265625
Epoch 5 mean of forward time is :0.12026917934417725
```

Βλέπουμε πως σε γενικές γραμμές είναι μεταξύ 4-6 φορές γρηγορότερη η διαδικασία αυτή όταν χρησιμοποιούμε το packing (γρηγορότερη για μεγαλύτερο batch size με βάση τα παραπάνω).

Ωστόσο αν τυπώσουμε τους χρόνους που χρειάστηκε να ολοκληρωθεί κάθε εποχή θα παρατηρήσουμε ότι έχουμε τους εξής χρόνους (αριστερά για 64 και δεξιά για 512 batch size) :

Epoch 1 time is:1.3298020362854004 Epoch 2 time is:1.3207051753997803 Epoch 3 time is:1.3177218437194824	Epoch 1 time is:2.2020978927612305 Epoch 2 time is:2.1773321628570557 Epoch 3 time is:2.1971747875213623
Epoch 1 time is:2.241380214691162 Epoch 2 time is:2.1931564807891846 Epoch 3 time is:2.225101947784424	Epoch 1 time is:1.2824106216430664 Epoch 2 time is:1.2633514404296875 Epoch 3 time is:1.2638614177703857

Όπου βλέπουμε πως για μικρό batch size έχουμε πράγματι βελτίωση στο συνολικό χρόνο εποχής ωστόσο αυξάνοντας το βλέπουμε να αντιστρέφονται τα αποτελέσματα και η packed λύση οδηγεί σε χειρότερο χρόνο.

Τέλος το παραπάνω παρατηρούμε ότι οφείλεται στην loss.backward διαδικασία η οποία στην περίπτωση packed για το πρώτο batch κάθε εποχής έχουμε μεγαλύτερο χρόνο για την διαδικασία αυτή , ειδικά στα μεγαλύτερα batch καθώς και λίγο μεγαλύτερο χρόνο και στα υπόλοιπα batch. Ωστόσο η αρχική αυτή τιμή είναι αρκετή στα μεγάλα batch sizes ώστε να οδηγήσει σε χειρότερο χρόνο εποχής όπως φαίνεται παρακάτω (Αριστερά packed):

```
Loss backward time is: 1.4791548252105713
Loss backward time is: 0.24724888801574707
Loss backward time is: 0.1718146800994873
Loss backward time is: 0.11801815032958984
Loss backward time is: 0.01565408706665039
Epoch 1 time is:2.215487241744995
```

```
Loss backward time is: 0.15046334266662598
Loss backward time is: 0.15546727180480957
Loss backward time is: 0.13860011100769043
Loss backward time is: 0.13937592506408691
Loss backward time is: 0.04520225524902344
Epoch 1 time is:1.301119089126587
```

Όπου παρατηρούμε το παραπάνω συμπέρασμα μας.
Το παραπάνω θεωρούμε οφείλεται (καθώς δεν γνωρίζουμε επακριβώς πως γίνεται η υλοποίηση από την pytorch και δεν είναι στα πλαίσια της εργασίας κάποια περαιτέρω ανάλυση) στο γεγονός πως το πρώτο πέρασμα κάθε εποχής η pytorch λογικά είναι απαραίτητη κάποια αρχικοποίηση για να εφαρμοστεί στην συνέχεια ο κανόνας αλυσίδας για το back propagation, ωστόσο λόγω του pack και ξανά pad οδηγούμαστε σε μετατροπές που αν και δεν αλλάζουν το αποτέλεσμα πιθανώς οδηγούνε τελικά σε πιο πολύπλοκη δομή.