Αναγνώριση Προτύπων
Αναφορά Προπαρασκευής
Εργαστήριο 1

Παναγιώτης Κωνσταντίνος Κακκαβάς AM 03120866

Κωνσταντίνος Μπάρκας AM 03120849

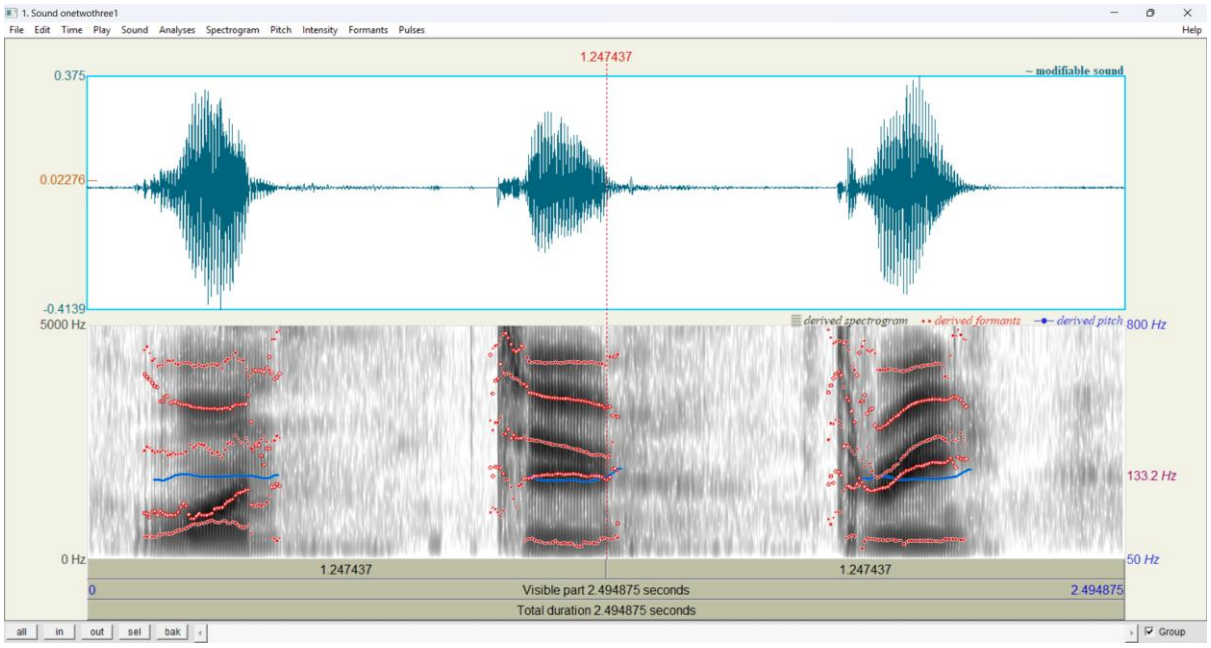
ΒΗΜΑ 1 PRAAT

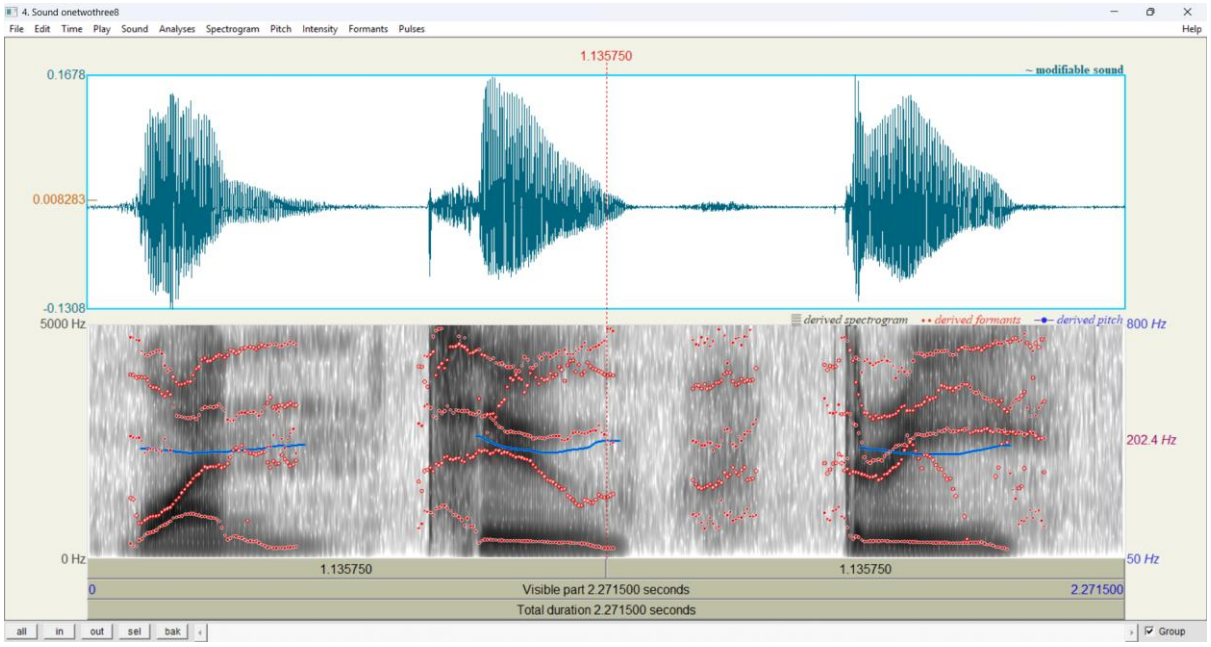
Εύκολα παρατηρούμε τα μοτίβα στα φωνήεντα τόσο στο πεδίο του χρόνου (άνω μισό του γραφήματος) όσο και στο πεδίο της συχνότητας (κάτω μισό του γραφήματος με μαύρο χρώμα). Επίσης παρατηρούμε την σταθερότητα του pitch για κάθε φώνημα ξεχωριστά.

HIGH LEVEL ΣΥΓΚΡΙΣΗ

Ήδη βλέποντας επιφανειακά τα δυο διαφορετικά αποτελέσματα παρατηρούμε ότι

- **Γυναίκα** (onetwothree8/κάτω) έχει **υψηλότερο pitch** από τον Άνδρα (onetwothree1/πάνω).
- Η καμπύλη του **pitch και στους δυο έχει παρόμοια μορφή** για τα αντίστοιχα φωνήματα.
- Το ίδιο συμβαίνει και με τα σημεία των formants (με σχετικά μικρές διαφοροποιήσεις σε κάποιες περιπτώσεις).
- Και στους δυο παρουσιάζονται **παρόμοια μοτίβα στα spectrograms** ανάλογα με το φώνημα που εκφέρουν.
- Η **ομοιότητα στις κυματομορφές** είναι εμφανής αλλά δεν φαίνεται να είναι αρκετή πληροφορία για να εξάγουμε συμπεράσματα.

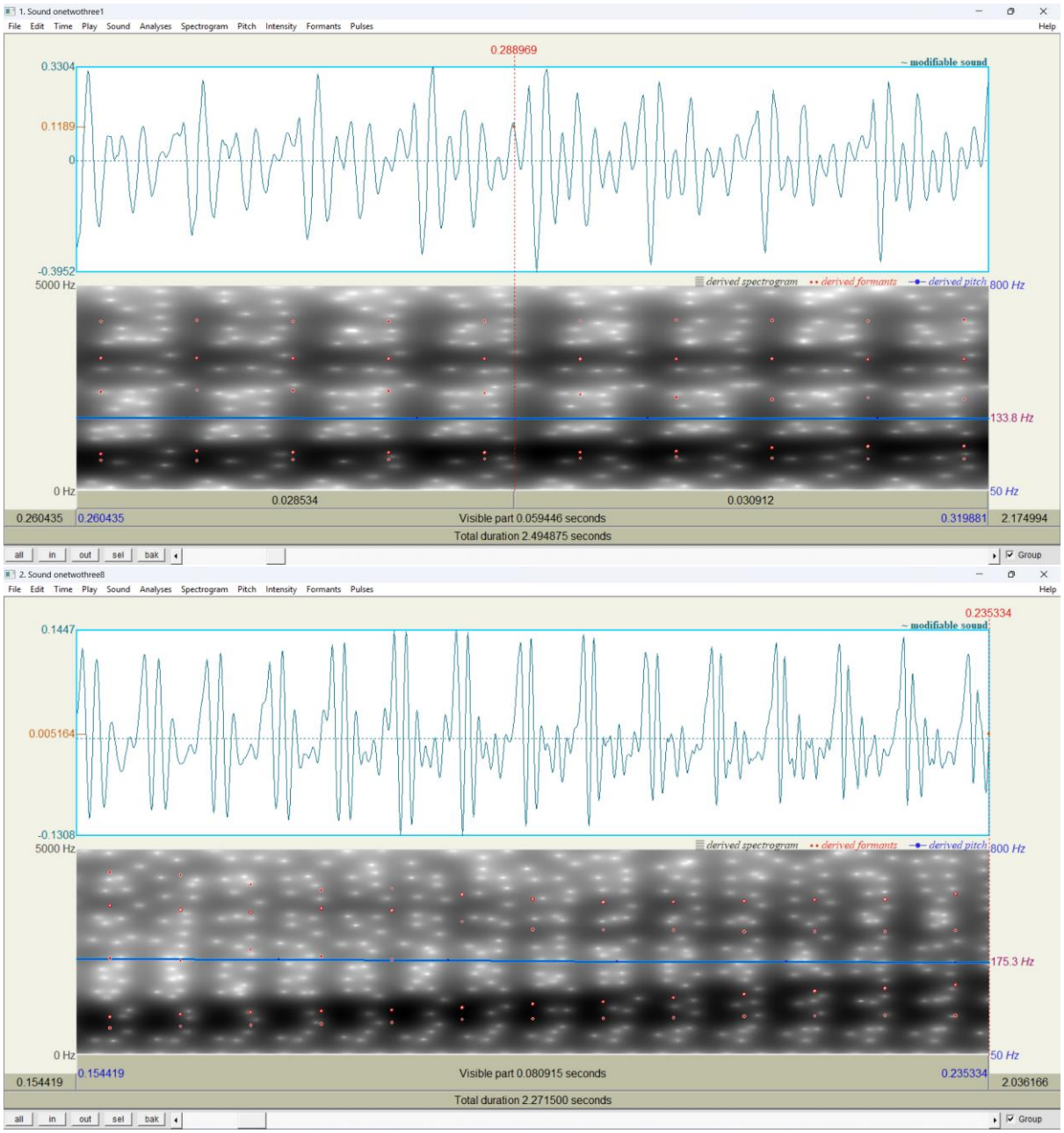




LOW LEVEL ΣΥΓΚΡΙΣΗ

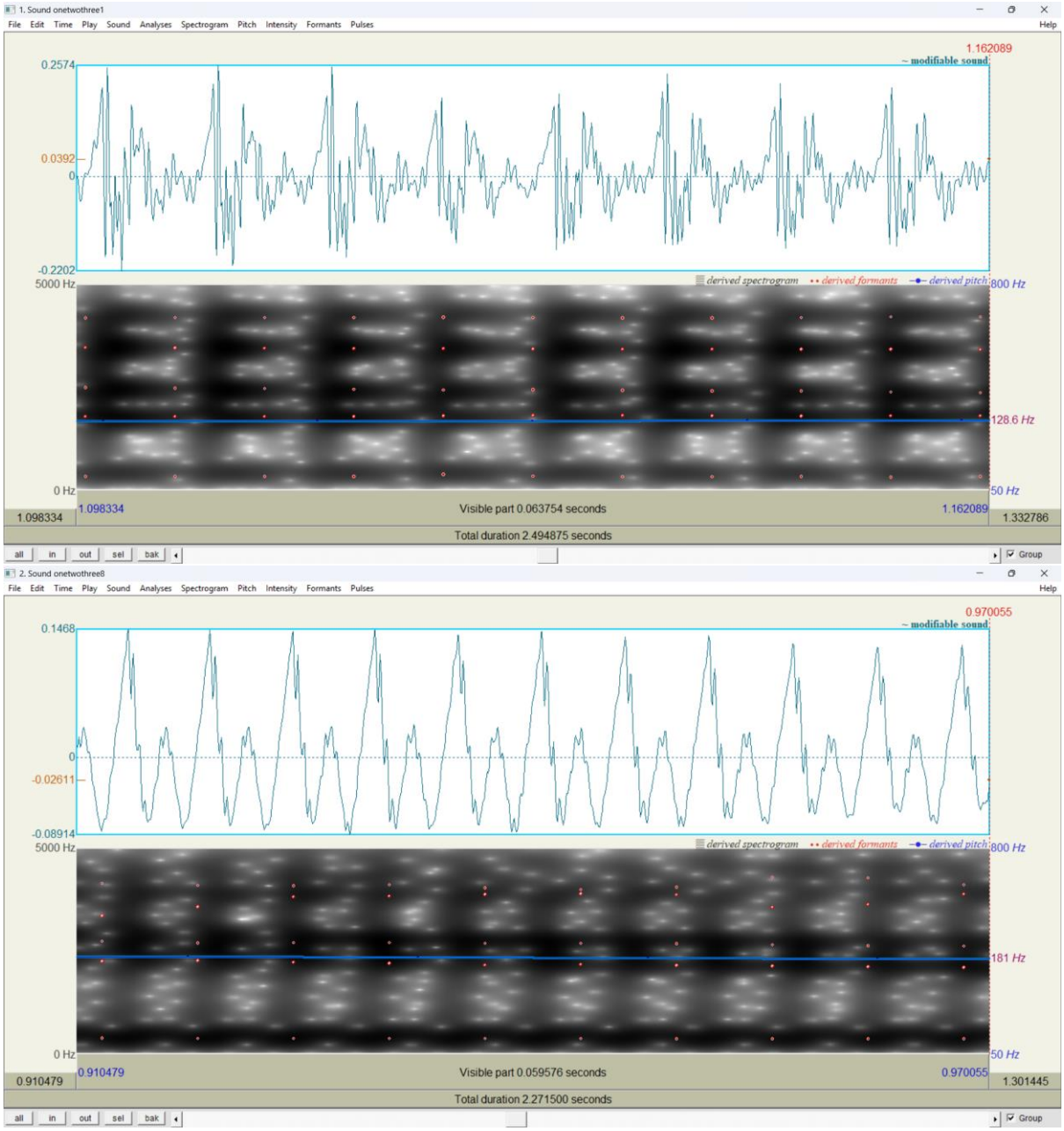
Φώνημα α

Παρατηρώ ότι το μοτίβο του “α” όπως φαίνεται στην εικόνα έχει μέση τιμή pitch τα 134 Hz και 175Hz για τα δυο δείγματα και παρουσιάζει πράγματι τις ομοιομορφίες που συζητήθηκαν παραπάνω.



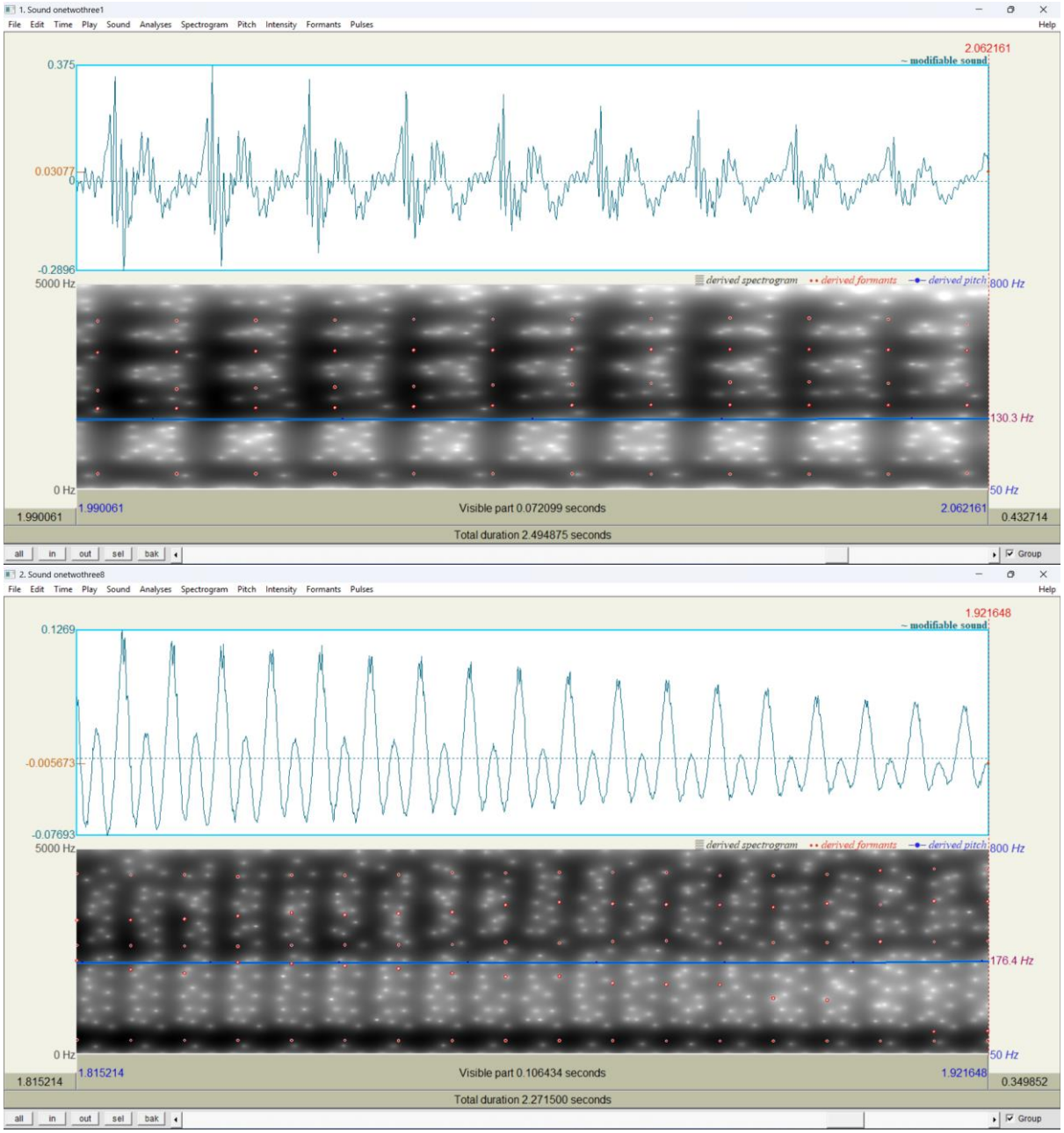
Φωνημα ου

Παρατηρώ ότι το μοτίβο του “ου” όπως φαίνεται στην εικόνα έχει μέση τιμή pitch τα 127 Hz και 181Hz για τα δυο δείγματα και παρουσιάζουν μεγάλη ομοιότητα στο spectrum και σχετικά μικρή στις απλές κυματομορφές.



Φωνημα ι

Παρατηρώ ότι το μοτίβο του “ι” όπως φαίνεται στην εικόνα έχει μέση τιμή pitch τα 130 Hz και 176Hz για τα δυο δείγματα και παρουσιάζει πράγματι τις ομοιομορφίες που συζητήθηκαν εξ αρχής.



ΒΗΜΑ 2

Απλή εξαγωγή με python.

ΒΗΜΑ 3

Υπολογισμός των delta.
Κατά τον υπολογισμό των τοπικών παραγώγων delta χρησιμοποιήθηκε η τεχνική

```
# function from librosa/feature/utils.py to calcite derivatives.
def delta(…):
   result: np.ndarray = scipy.signal.savgol_filter(
      data, width, deriv=order, axis=axis, mode=mode, **kwargs
   )
   return result
```

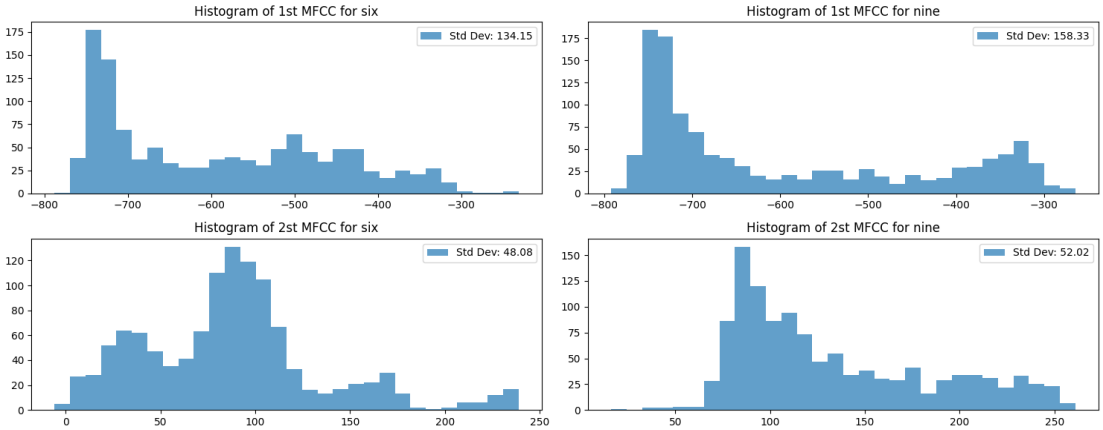
Φιλτράρισμα Savitzky-Golay:

- Το φιλτράρισμα Savitzky-Golay είναι μια τεχνική που εφαρμόζει ένα φίλτρο χαμηλής διέλευσης για να εξομαλύνει το σήμα, ενώ επιτρέπει και τον υπολογισμό της παραγώγου του.
- Λειτουργεί προσαρμόζοντας ένα πολυώνυμο σε κάθε τοπικό παράθυρο σημείων και στη συνέχεια υπολογίζει την παράγωγο αυτού του πολυωνύμου στο κεντρικό σημείο του παραθύρου.
- Το φίλτρο αυτό παρέχει μια ομαλή εκτίμηση της παραγώγου, κάνοντάς το μια δημοφιλή επιλογή για υπολογισμό των χαρακτηριστικών διαφοράς.

Υπολογισμός του MFCC:
Η διαδικασία για το Mel Frequency Cepstral Coefficients (MFCC) Explained είναι παρόμοια με την εξαγωγή του Cepstrum με την προσθήκη μιας συνθήκης πριν την λογαρίθμιση του μετασχηματισμού Fourier του σήματος όπου το μετατρέπουμε σε Mel scale όπου ένα mel ισούται με $m = 2595log_{10}\left(1 + \frac{f}{700}\right)$, η νέα αυτή κλίμακα διευκολύνει τον διαχωρισμό των λέξεων ήχων με βάση το pitch με σκοπό να ισαπέχουν τα διαφορετικά pitch μεταξύ τους.

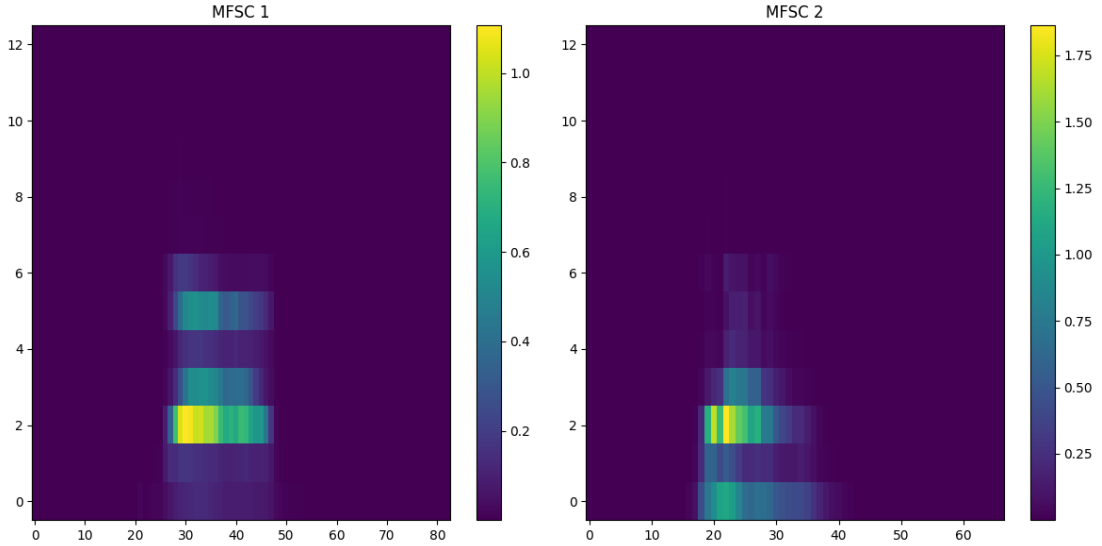
Για αυτό και επιλέγουμε τα δείγματα μας για το επόμενο με ίσες αποστάσεις αλλά για μειώσουμε το πιθανό λάθος από μικρό παράθυρο και παράλειψη πιθανώς χρήσιμης γειτονικής πληροφορίες σε κάθε σημείο που δεχόμαστε εφαρμόζουμε ένα τριγωνικό φίλτρο σαν weighted average για να μειώσουμε το παραπάνω σφάλμα.

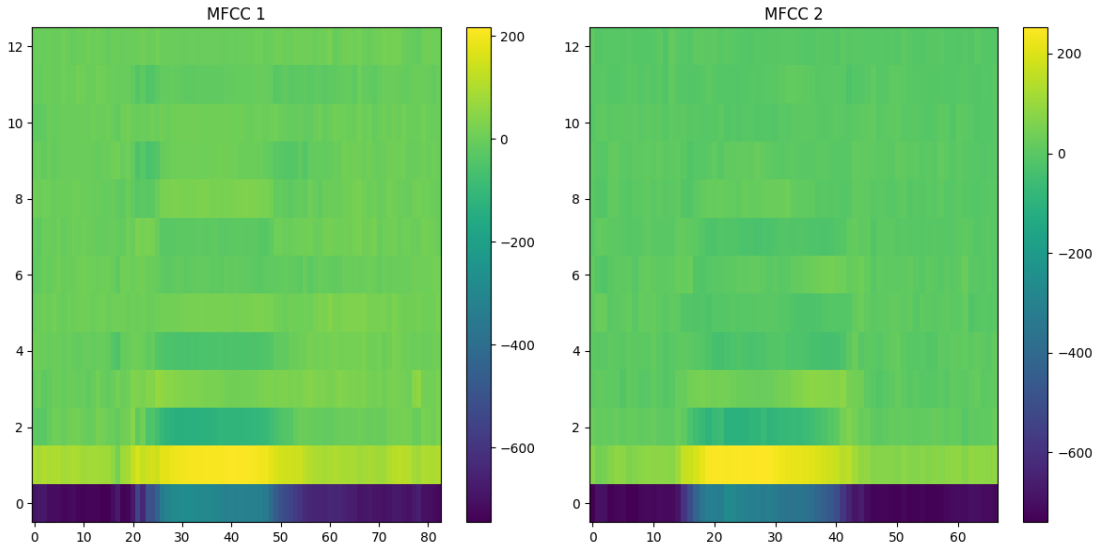
ΒΗΜΑ 4



Για κάθε διαφορετική εκφώνηση ψηφίου έχουν εξαχθεί vectors διάστασης 13 μεταβλητού πλήθους (ίσο με το πλήθος των παραθύρων που χρησιμοποιήθηκαν στην συγκεκριμένη εκφώνηση).

Παρατηρώντας τις τυπικές αποκλίσεις (βλ. τα labels των histograms) προβλέπουμε ότι και στα δύο ψηφία ο 1ος συντελεστής MFCC έχει αρκετή απόκλιση, ενώ για τον 2ο και 3ο συντελεστή η απόκλιση μειώνεται σημαντικά.





Οι συντελεστές MFSC παρουσιάζουν συσχέτιση, η οποία μειώνεται στους MFCCs λόγω του DCT ο οποίος προκαλεί κάτι σαν αποσυσχέτιση/συμπίεση (για αυτό και οι δεύτεροι είναι προτιμότεροι). Για τον υπολογισμό της συσχέτισης υπολογίζουμε την αυτοσυσχέτιση των συντελεστών (άξονας y στα διαγράμματα) για κάθε χρονικό παράθυρο και μετά παίρνουμε την μέση τιμή όλων αυτών των αυτοσυσχετίσεων.

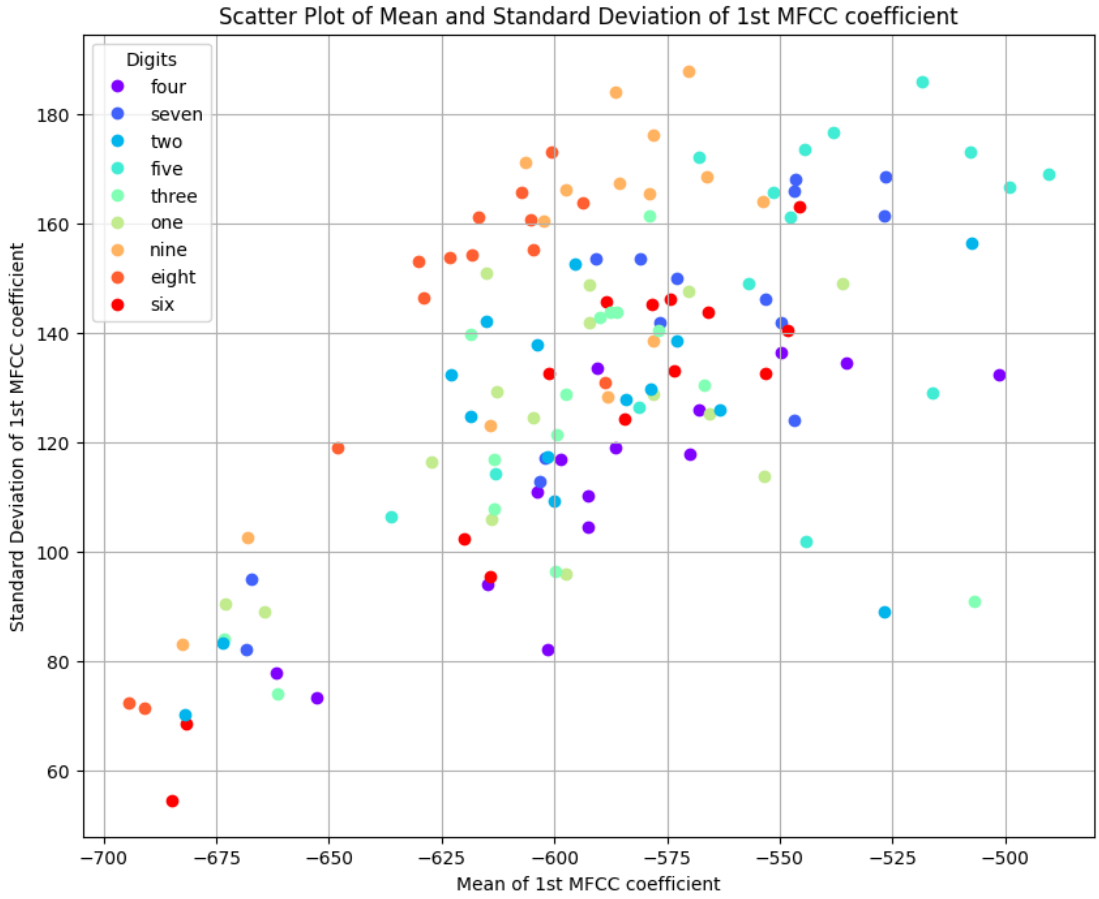
Mean Autocorrelation for MFSC Features: [0.03846154 0.03846153]

Mean Autocorrelation for MFCC Features: [0.01282051 0.01282051]

ΒΗΜΑ 5

Για κάθε ένα από τα 39 χαρακτηριστικά (13 συντελεστές, 13 deltas, 13 delta-deltas) δημιουργούμε 2 στοιχεία (μέση τιμή και τυπική απόκλιση), οπότε το τελικό διάνυσμα που αντιπροσωπεύει την εκφώνηση έχει διάσταση 39*2 = 78.

Παίρνουμε το scatter plot για τις 2 πρώτες διαστάσεις:



Παρατηρούμε ότι λαμβάνοντας υπόψη μόνο τα 2 πρώτα στοιχεία του τελικού vector (μέση τιμή και τυπική απόκλιση του 1ου συντελεστή MFCC) δεν μπορούμε ξεκάθαρα να διακρίνουμε τις κλάσεις των διαφορετικών ψηφίων (μπορεί να γίνει μόνο ένας μερικός διαχωρισμός ανάμεσα σε συγκεκριμένα ψηφία τα οποία διαχωρίζονται καλά με βάση τα 2 αυτά χαρακτηριστικά, π.χ. five vs three).

ΒΗΜΑ 6

Percentage of variance preserved with 2 dimensions: 70.65%

Βλέπουμε ότι παρόλο που και πάλι κρατάμε 2 διαστάσεις, οι 2 αυτές διαστάσεις της PCA είναι πολύ πιο «χρήσιμες» σε σχέση με τις 2 πρώτες που πήραμε αυθαίρετα στο προηγούμενο βήμα, για αυτό και επιτυγχάνουν ελαφρώς καλύτερο διαχωρισμό μεταξύ των κλάσεων. Ωστόσο βλέπουμε κιόλας ότι το ποσοστό της διασποράς που διατηρείται είναι σχετικά μικρό με μόνο 2 διαστάσεις (όχι ικανοποιητική PCA).

Percentage of variance preserved with 3 dimensions: 81.49%

Με 3 διαστάσεις βλέπουμε ότι το ποσοστό της διατηρούμενης διασποράς είναι σαφώς μεγαλύτερο, γεγονός που αποτυπώνεται και στο scatter plot όπου ο διαχωρισμός των κλάσεων (ψηφίων) είναι σαφώς πιο ξεκάθαρος.

ΒΗΜΑ 7

Τα μοντέλα μας είναι τα εξής: Naive Bayes, SVM, Decision Tree και MLP 2 στρωμάτων. Η εκπαίδευση-αξιολόγηση των μοντέλων έγινε με dataset το οποίο προήλθε από το αρχικό (78 διαστάσεις) έπειτα από PCA η οποία κράτησε επαρκείς διαστάσεις για την διατήρηση του 95% του variance.

Explained variance ratio with 9 components: 95.90%

Αρχικά (κατά λάθος) έγινε εκπαίδευση χωρίς να προηγηθεί κανονικοποίηση, με τις επιδόσεις να είναι οι εξής:

Naive Bayes accuracy: 62.50%

SVM accuracy: 72.50%

Decision Tree accuracy: 37.50%

MLP accuracy: 72.50%

Η ίδια ακριβώς διαδικασία με κανονικοποίηση αυξάνει φανερά τις επιδόσεις των (περισσότερων) μοντέλων μας:

Naive Bayes accuracy: 62.50%

SVM accuracy: 80.00%

Decision Tree accuracy: 45.00%

MLP accuracy: 85.00%

ΒΗΜΑ 8

Εκπαίδευση δικτύου έγινε ένα LSTM με δοκιμή των υπερ-παραμέτρων {segment_size, hidden_size, num_layers, batch_size και epochs, στα 4/5 της περιόδου του ημιτόνου. Χρησιμοποιήθηκαν τα ελάχιστα τετράγωνα σαν σφάλμα και ο Adam optimizer.

Τα δύο LSTM που χρησιμοποιήθηκαν παρουσιάζονται παρακάτω:

Χρήση torch.nn.LSTM()

```
class SimpleLSTM(nn.Module):
    def __init__(self, input_size=1, hidden_size=config['hidden_size'], num_layers=config['num_layers']):
        super(SimpleLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)  # or use the custom one
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out)
        return out
```

Χρήση MultilayerCustomLSTM() (μια αυτοσχέδια κλάση που υλοποιεί την λογική του LSTM όπως παρουσιάζεται παρακάτω

```
class MultiLayerCustomLSTM(nn.Module):  6 usages  new *
    def __init__(self, input_size, hidden_size, num_layers=1, dropout=0.0):  new *
        super(MultiLayerCustomLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # Use ModuleDict to store each layer of CustomLSTM
        self.layers = nn.ModuleDict({
            f'lstm_layer_{i}': CustomLSTM(input_size if i == 0 else hidden_size, hidden_size,
                                          dropout=dropout if i < num_layers - 1 else 0.0)
            for i in range(num_layers)
        })

    def forward(self, x, init_states=None):  new *
        layer_states = []
        if init_states is None:
            init_states = [(torch.zeros(x.size(0), self.hidden_size).to(x.device),
                            torch.zeros(x.size(0), self.hidden_size).to(x.device))
                           for _ in range(self.num_layers)]

        # Pass through each layer in sequence
        for i in range(self.num_layers):
            layer = self.layers[f'lstm_layer_{i}']
            x, (h_t, c_t) = layer(x, init_states[i])
            layer_states.append((h_t, c_t))  # Store hidden and cell states

        # Return the output from the final layer and states for all layers
        return x, layer_states
```

Δομή εξόδου: Το δίκτυο παράγει εξόδους σε κάθε χρονικό βήμα και έχει αναδρομικές συνδέσεις μεταξύ των κρυφών του μονάδων.

Επιλογή αρχιτεκτονικής: Στο παράδειγμα χρησιμοποιούμε LSTM καθώς έχουν δείξει καλύτερα αποτελέσματα σε πολύ μεγάλη πληθώρα προβλημάτων σε σχέση με την κλασική προσέγγιση.

Επικύρωση:

Έγιναν δυο δοκιμές:

1. Πρόβλεψη **ένα** σημείο τη φορά (**πορτοκαλί** σε Figure 1, Figure 2)
2. Πρόβλεψη **πολλαπλών** σημείων τη φορά (**μπλε** σε Figure 1, Figure 2)
    a. Το σφάλμα σε κάθε πρόβλεψη πολλαπλασιάζεται για αυτό παρατηρούμε και αυτή την ξαφνική απόκλιση μετά από μερικά βήματα στο μοντέλο.
    b. Με αυτή τη δοκιμή δεν βλέπουμε μόνο την ακρίβεια στις προβλέψεις του στο επόμενο βήμα αλλά μας βοηθάει να δούμε και το πόσο καλά γενικεύει και πόσο ικανό ήταν πράγματι να καταλαβαίνει πλέον μόνο του την ακολουθία βημάτων για την αναδημιουργία της παραγώγου.

Γραφήματα: Σε αυτή την περίπτωση κάνουμε Κανένα σύνολο ύπερ-παραμέτρων δεν έλυσε το πρόβλημα. Παρατηρούμε ότι σε αρκετές περιπτώσεις το μοντέλο μαθαίνει αρκετά καλά το train-set (80% of sin) και όπως και αναμέναμε έχει μεγάλη απώλεια ακρίβειας στο dataset.

Συμπεράσματα για τις υπερ-παραμέτρους

1. **Μήκος Ακολουθίας**: Μεγαλύτερο μήκος ακολουθίας βοηθά στην ομαλή πρόβλεψη, ενώ μικρότερο μπορεί να οδηγήσει σε αστάθεια και απώλεια σημαντικών εξαρτήσεων.
2. **Μέγεθος Κρυφής Μνήμης**: Μεγαλύτερο μέγεθος κρυφής μνήμης επιτρέπει την εκμάθηση πιο πολύπλοκων μοτίβων, αλλά μπορεί να οδηγήσει σε υπερπροσαρμογή.
3. **Μέγεθος Παρτίδας (Batch Size)**: Μικρότερο μέγεθος παρτίδας προσθέτει θόρυβο και αργή σύγκλιση, ενώ μεγαλύτερο οδηγεί σε πιο σταθερές προβλέψεις.
4. **Αριθμός Επιπέδων**: Περισσότερα επίπεδα δίνουν μεγαλύτερη δυνατότητα μάθησης αλλά δυσκολεύουν την εκπαίδευση, ενώ λιγότερα επίπεδα απλοποιούν το μοντέλο.
5. **Εποχές**: Περισσότερες εποχές βελτιώνουν την ακρίβεια, αλλά χωρίς ρύθμιση μπορεί να οδηγήσουν σε υπερπροσαρμογή, ενώ λιγότερες εποχές αφήνουν την εκπαίδευση ελλιπή.

Εμπειρικά Συμπεράσματα:

- Το μήκος ακολουθίας και το μέγεθος κρυφής μνήμης επηρεάζουν σημαντικά την απόδοση του μοντέλου.
- Μέτριο μέγεθος κρυφής μνήμης και κατάλληλο μήκος ακολουθίας οδηγούν σε πιο σταθερή απόδοση χωρίς σημαντικές διακυμάνσεις.

Σύγκριση μοντέλων: Επιστρέψανε πολύ παρόμοια αποτελέσματα. Η μόνη διαφορά που παρατηρούμε είναι ότι το Custom όταν κάνει segment prediction, δεν έχει τόσο απότομα σφάλματα. Οι δυο εικόνες με τα γραφήματα για κάθε μοντέλο παρουσιάζονται παρκάτω.

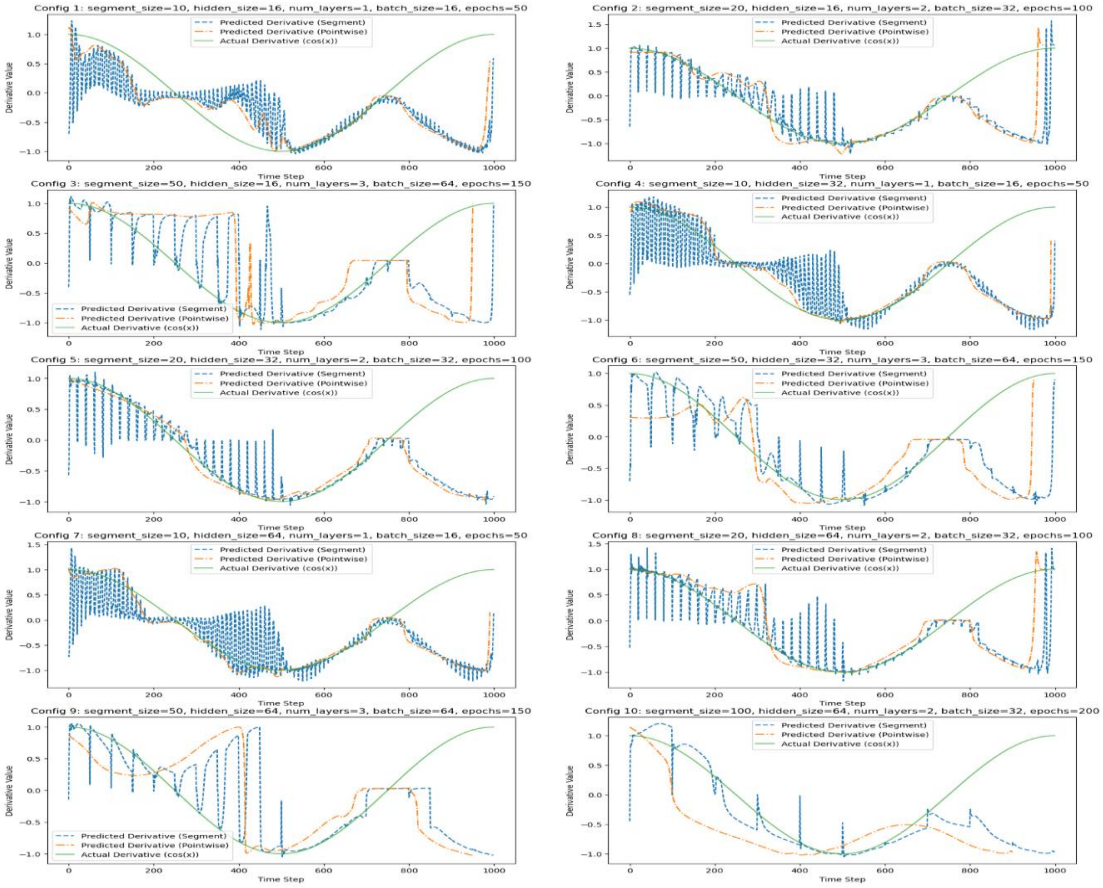


Figure 1 Αποτελέσματα εκπαίδευσης του torch.nn.LSTM

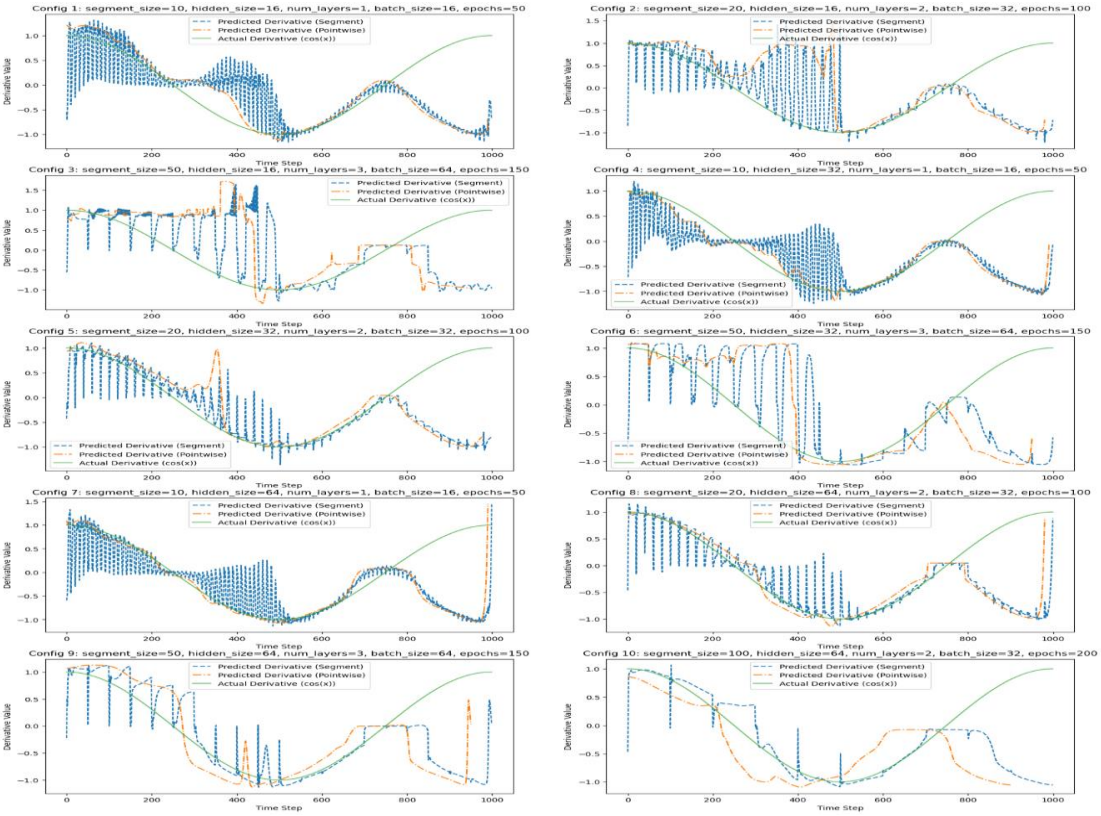


Figure 2 Αποτελέσματα εκπαίδευσης του MultilayerCustomLSTM