ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

Sign Language Recognition in Video Sequences of Single Words

by

Konstantinos Nikolopoulos

Submitted

in partial fulfilment of the requirements for the degree of

Master of Big Data & Analytics

at the

UNIVERSITY OF PIRAEUS

February 2024

Thesis Supervisor: Ilias Maglogiannis

Title: Dean of School

Author: Konstantinos Nikolopoulos

Abstract

The goal of this thesis is to explore the challenges of the Sign Language Recognition (SLR) problem, and suggest an accurate Machine Learning (ML) model for SLR in video sequences of single words. SLR holds significant importance as it addresses the communication barriers between individuals with hearing impairments or speech impediments and the general population. However, the existing methods face various constraints. Many proposed solutions rely on image-based recognition, while others require the use of multi-colored/sensor-based gloves or specific cameras. This study proposes a straightforward system that does not require specific accessories, yet remains highly resilient to variations in test subjects such as skin tone, gender, and body size. This signer-independent system consists of four main steps. Firstly, a dataset was gathered for three target corpus sizes (20, 100 and 300 words) that is both balanced and with high variability. For that reason, the "WLASL: A large-scale dataset for Word-Level American Sign Language" was selected. Then arm and hand features were extracted from the videos using real-time optimized Computer Vision libraries, frameworks and Machine Learning (ML) solutions. The tools of choice are mainly Mediapipe and OpenCV. Afterwards, data augmentation and dynamic time wrapping techniques were applied to the data to improve performance and invariance. Finaly, a selection of Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and combinations of the two were trained. The experiments showed that the proposed approach wields excellent results especially for the CNN models, reaching up to 98% accuracy for a corpus size of 100 words or 97% for a corpus size of 300 words.

Περίληψη

Στόχος αυτής της διατριβής είναι να διερευνήσει τις προκλήσεις του προβλήματος Αναγνώρισης Χειρονομιών Νοηματικής Γλώσσας, και να προτείνει ένα ακριβές μοντέλο Μηχανικής Μάθησης για την Αναγνώριση Χειρονομιών Νοηματικής σε ακολουθίες βίντεο μεμονωμένων λέξεων. Η Αναγνώριση Χειρονομιών Νοηματικής έχει ιδιαίτερη σημασία καθώς αντιμετωπίζει τα εμπόδια επικοινωνίας μεταξύ ατόμων με αναπηρία ακοής ή δυσκολίες στο λόγο και τον γενικό πληθυσμό. Ωστόσο, οι υπάρχουσες μέθοδοι αντιμετωπίζουν αρκετούς περιορισμούς. Πολλές από τις προτεινόμενες λύσεις βασίζονται σε τεχνικές αναγνώρισης εικόνας, ενώ άλλες απαιτούν τη χρήση γαντιών πολλαπλών χρωμάτων/αισθητήρων ή καμερών με συγκεκριμένες προδιαγραφές. Η μελέτη αυτή προτείνει ένα απλό σύστημα που δεν απαιτεί εξειδικευμένο εξοπλισμό, αλλά παραμένει ανθεκτικό σε διακυμάνσεις των χαρακτηριστικών των ομιλητών, όπως η απόχρωση του δέρματος, το φύλο και το μέγεθος του σώματος. Το σύστημα αυτό, που δεν εξαρτάται από τον ομιλητή, αποτελείται από τέσσερα κύρια βήματα. Αρχικά, συλλέχθηκε ένα σύνολο δεδομένων, και δημιουργήθηκαν τρεις συλλογές λέξεων (20, 100 και 300 λέξεις) που είναι ισορροπημένο και με υψηλή ποικιλομορφία. Για αυτόν τον λόγο, επιλέχθηκε το "WLASL: A large-scale dataset for Word-Level American Sign Language ". Στη συνέχεια, εξήχθησαν χαρακτηριστικά χεριών από τα βίντεο χρησιμοποιώντας βιβλιοθήκες πραγματικού χρόνου βελτιστοποιημένης Υπολογιστικής Όρασης, και μεθόδων Μηχανικής Μάθησης. Τα εργαλεία που επιλέχθηκαν είναι κυρίως το Mediapipe και το OpenCV. Έπειτα, εφαρμόστηκαν τεχνικές data augmentation και αντιστοίχισης δυναμικού χρόνου (Dynamic Time Wrapping) στα δεδομένα για να βελτιωθεί η απόδοσή του. Τέλος, εκπαιδεύτηκαν μοντέλα Συνελικτικών Νευρωνικών Δικτύων (CNN), Αναδρομικών Νευρωνικών Δικτύων (RNN) και συνδυασμοί των δύο. Τα πειράματα έδειξαν ότι η προτεινόμενη προσέγγιση προσφέρει εξαιρετικά αποτελέσματα, ειδικά για τα μοντέλα CNN, φτάνοντας έως και 98% ακρίβεια για ένα σύνολο δεδομένων 100 λέξεων ή 97% για ένα σύνολο δεδομένων 300 λέξεων.

Contents

List of Figures

List of Tables

1. Introduction

Speech is the cornerstone of human communication. With communication information is exchanged between individuals. In order for the communication between two parties to be effective, both parties should be able to comprehend and acknowledge the conveyed messages. There are about 75 million Deaf people worldwide [1, 2, 3]. Most of these people use sign languages as their primary or only means of communication. A communication gap exists when a Deaf person and a hearing person attempt to communicate [2].

Despite common belief, sign languages are not the corresponding translation in signs of spoken languages. Sign languages are fully fledged languages with unique grammar and syntax. They were developed within Deaf communities to satisfy the need to communicate with each other with minimal influence from the spoken languages. Another misconception is that sign language is a single universal language. In reality, most countries have their own distinct sign language. For example, in the United States of America the Deaf communicate in American Sign Language and in Greece the Deaf communicate in Greek Sign Language. These languages may have a few common signs, but they are mostly distinct. Other examples of sign languages that are similarly distinct include the British Sign Language, Japanese Sign Language, German Sign Language, South African Sign Language, each unique to a specific country, and generally distinct from each other. Moreover, it is incorrectly assumed that the Deaf can read and write spoken languages. The primary language amongst the Deaf in Greece is Greek Sign Language (GSL). The ability for the Deaf to read and write in spoken languages requires special training, which may be expensive and unavailable to most of the Greek Deaf. State resources are limited, and skilled interpreters are scarce and their use has been limited to official applications rather than everyday common use.

One of the main goals of the project is to make use of inexpensive commodity hardware such as monocular cameras, making use of computer vision to extract sign language semantic information from 2D video, instead of using expensive hardware placed over the body to track the user’s hands and body.

1.1. Sign language basic structure

Sign language includes different components of visual actions of the signer made by using the hands, the face, and the arms/torso, to convey the meaning.

1.1.1. Manual components

In sign language the information is mainly expressed by hand/arm gestures with various positions, orientation, configurations and movement of the hands. [Figure 1-1] shows sequential frames of the sign language sentence "WOMAN ARRIVE HERE" from the RWTH-BOSTON-104 dataset. Such gestures are called manual components of the signing and convey the main meaning of the sign. Manual components are integral, and are divided into two categories; glosses and classifiers.

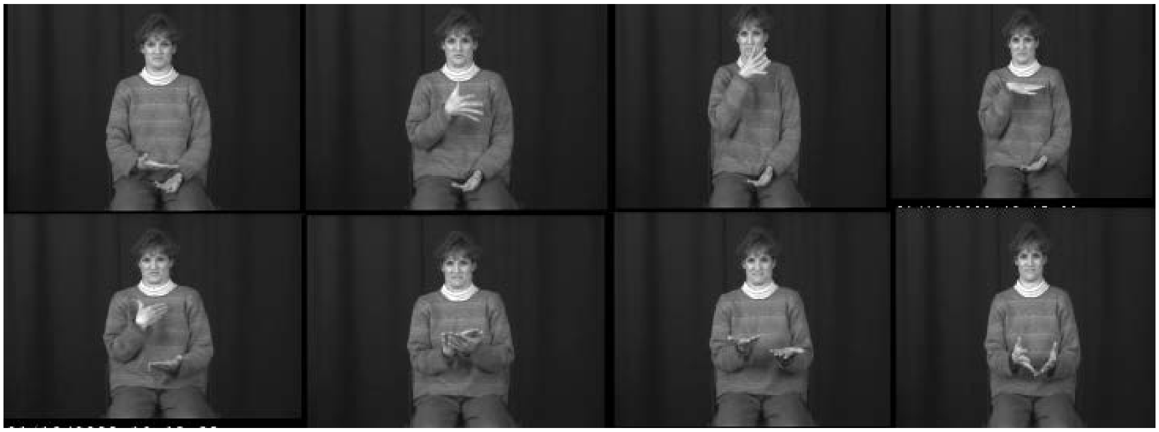


Figure 1-1 The key-point frames of the phrase "Woman Arrived Here" in ASL from the RWTH-BOSTON-104 dataset

Glosses are the actual signs that are performed for a sign language word like dog or cat. On the other hand, classifiers are used in American Sign Language (ASL) to express movement, location, and appearance of a person, a subject, or an object. Signers use classifiers as nouns or pronouns to express a sign language word by describing its location and its movement, or its appearance. In a continuous signing of a sentence, not only the hand(s) moves from the ending position of one sign to the starting position of the next, but the hand(s) configuration also changes. This is what is called movement epenthesis and even though it happens with every transition of a sign to another, it does not belong to the set of components of sign language. The fourth frame in [Figure 1-1] shows a frame of movement epenthesis which occurs between the sign “WOMAN” and the sign “ARRIVE”.

1.1.2. Non-manual components

The manual components of sign language are fundamental, so big part of the research in the field of sign language recognition focuses on their recognition. However, the non-manual components of sign language play a significant role also. These signals are conveyed through facial expressions such as eye gaze, movement of eyebrows, head movement and posture, and movements of signers’ torso. In [Figure 1-1], for example, the signer moves the lips wide in the second and third frame in parallel of signing the manual component “WOMAN”. To effectively construct a system which is able to understand sign language, an analysis of non-manual signs and lexicons is required.

1.1.3. Grammar

There is no particular similarity between the grammar of a region’s sign language and that of a spoken one, even though sign language follows a number of general rules of spoken language. For instance, the spoken language has a specific syntax with nouns, pronouns, verbs and adjectives in a specific order. The same does not apply in sign language. In sign language there are words in a spoken language sentence that are ignored in the translation. In addition, there are changes in the sign appearance during continuous signing which affect the sign meaning. In particular, these changes concern the sign language grammar, and are not reflected in the spoken language. As previously stated, signers utilize multiple body parts such as hands, head, face and overall body configuration to perform a sign sentence. Consequently, sub-lexical components may be simultaneously signed by various parts of the body. This fundamental characteristic distinguishes sign language grammar from that of spoken language.

Another unique characteristic of sign language grammar is indexing. The signer identifies individuals and objects by employing classifiers or by spelling their names using the sign language alphabet, positioning them within the signing space around themselves. Subsequently, they can easily reference them by simply pointing to the previously designated location. This indexing technique also serves as a form of person agreement, distinguishing between first person, second person, or third person references. Additionally, when signing a verb, the direction of movement conveys information about both the subject and object of the action, through variations in starting and ending locations and hand orientation. Various methods of signing a verb can accommodate different types of agreements, indicating the number of individuals performing the action, the number of repetitions, or the number of objects or subjects involved. Emphatic inflections, used for emphasis, the derivation of nouns from verbs, numerical incorporation, and compound signs are further aspects of sign language grammar, resulting in systematic alterations.

1.2. Sign Language Notation Systems

Sign language diverges from spoken language in writing. Spoken languages are predominantly sequential, where phonemes form words, words construct sentences, and sentences comprise texts, all produced in a linear sequence. Thus, spoken languages can be written in sequence. However, sign language consists of manual and non-manual signs simultaneously executed by the hands, face, or head. As a result, unlike traditional writing systems, a notation system for sign language must capture the non-sequential nature of signs.

While some notation systems have been devised and are employed within small communities, researchers typically utilize the following five unofficial notation systems:

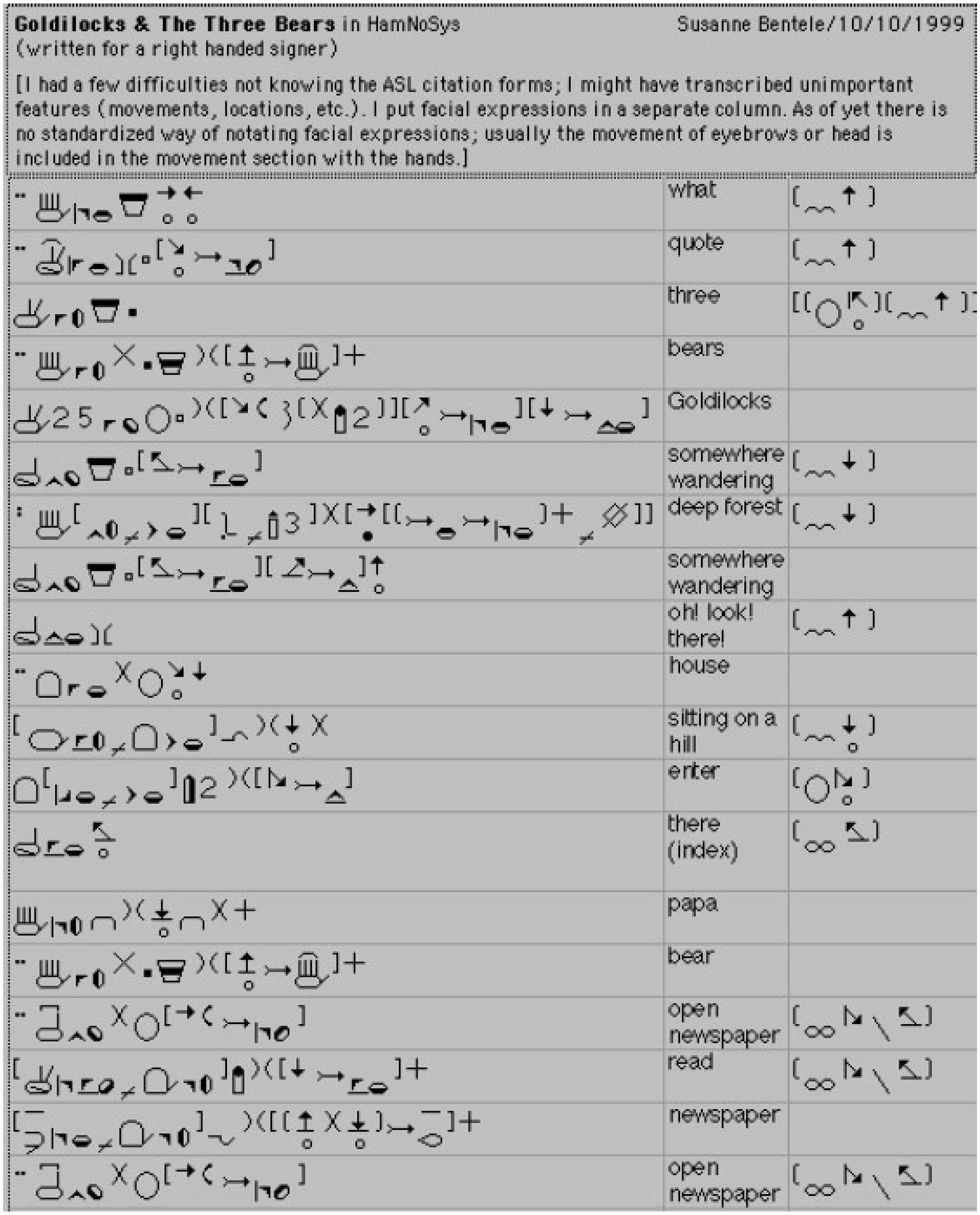
Glosses. Glosses serve as written representations of sign language words, offering a visual depiction of a sign. Each sign is substituted with a corresponding word in spoken language, with one sign replaced by one gloss. However, as glosses are not comprehensive translations of signs, additional transcriptions may include non-manual signs and classifiers alongside the main text. Glosses are commonly written in capital letters.

Stokoe system. The Stokoe system, named after William Stokoe, who pioneered its development, brought sign language to the attention of linguists by introducing this notation system. It stands as the first phonological symbol system designed to represent the constituent elements of American Sign Language (ASL). Comprising 55 symbols grouped into three categories—location (tab), hand shape (dez), and movement (sig)—these symbols are utilized simultaneously. The location and movement symbols are iconic, while hand shapes are represented by units derived from the number system and manual alphabet of ASL. Over time, various research teams have modified this notation to suit their specific needs, leading to the emergence of multiple variations. Consequently, the Stokoe system should be understood not as a singular notation but as a diverse family of related systems.

HamNoSys. The Hamburg Sign Language Notation System (HamNoSys), S. Prillwitz [4], serves as a "phonetic" transcription system, offering a broader scope than the Stokoe system. It consists of approximately 200 symbols utilized to transcribe signs across four levels: hand shape, hand configuration, location, and movement. These symbols are designed to be easily recognizable, drawing upon iconic representations. While the system also aims to incorporate facial expressions, its development in this aspect remains ongoing. HamNoSys continues to evolve and grow as the need dictates. [Figure 1-2] illustrates examples of sign language sentences in HamNoSys symbols.

SignWritting. SignWriting was introduced by Valerie Sutton in 1974 as an adaptation of a movement writing system known as Sutton "DanceWriting." This innovative system was developed to capture the intricate movements of sign languages. Sutton's groundbreaking work led to the publication of the first newspaper in history written in sign language using this notation system. However, it's important to note that Sutton herself did not have proficiency in the sign languages she transcribed. Consequently, SignWriting is unique in that it is not derived from or connected to any existing writing system. Instead, it records movements in a generic format, solely focusing on depicting the visual appearance of body movements. This universal approach enables SignWriting to represent any sign language worldwide, including detailed facial expressions and mimetic actions. [Figure 1-3] illustrates the sign for "quote" in various notation systems, including Glosses, Stokoe, HamNoSys, and SignWriting.

Liddell and Johnson. The Movement-Hold notation system was developed by Scott Liddell and Robert Johnson [5]. Unlike the graphical symbols used in the Stokoe model, this system employs English words to describe the physical actions involved in signing. In the Movement-Hold model, signing is divided into sequential segments known as movements and holds. Movements represent the parts of the signing process where sign changes, while holds represent segments where all aspects remain consistent for a minimum of 1/10 second. All the necessary information required to describe a sign are in these segments, including hand shapes, locations, orientations, as well as non-manual components such as facial expressions and torso configuration.



Goldilocks & The Three Bears in HamNoSys Susanne Bentele/10/10/1999
(written for a right handed signer)

[I had a few difficulties not knowing the ASL citation forms; I might have transcribed unimportant features (movements, locations, etc.). I put facial expressions in a separate column. As of yet there is no standardized way of notating facial expressions; usually the movement of eyebrows or head is included in the movement section with the hands.]

what
quote
three
bears
Goldilocks
somewhere wandering
deep forest
somewhere wandering
oh! look! there!
house
sitting on a hill
enter
there (index)
papa
bear
open newspaper
read
newspaper
open newspaper

Figure 1-2 Sample of sentences represented in the HamNoSys notation system. The HamNoSys symbols, and other corresponding meaning in English and facial expressions are ordered from the left to the right column. Source: http://www.signwriting.org/forums/linguistics/ling007.html

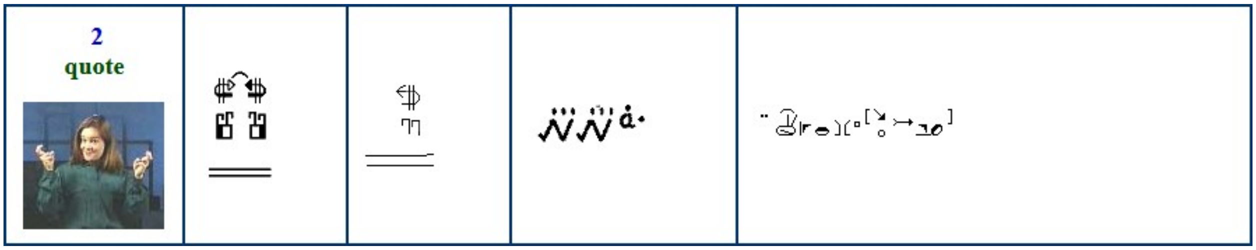


Figure 1-3 Sample of the sign "quote" presented in different systems; (a) shows the signing and (b), (c), (d) and (e) present it in Glosses, Stokoe, HamNoSys and SignWriting notation systems, respectively.
Source: http://www.signwriting.org/forums/linguistics/ling001.html

As previously noted, a sign language recognition system is the foundation for a communication system between the deaf and the hearing. This system typically consists of some hardware for data collection and feature extraction, and a decision-making component for sign language recognition. Many researchers use special hardware like data gloves, colored gloves, wearable cameras, or location sensors, in order perform feature extraction. However, the proposed system takes a different approach by aiming to recognize sign language words using landmark features of the signer, extracted directly from frames captured by standard cameras. Therefore, this system can be employed with ease in different environments. Even though we avoid many image-recognition issues when using landmark features, the sign language recognition system has to overcome other problems like visual variability of utterances of each sign language word, different pronunciations, and large amount of features extracted directly from the image frames. To address these issues, we are introducing an American Sign Language Recognition (ASLR) system which is based on existing lightweight Machine Learning models for hand, face and pose landmark detection; in this project we selected Google's Mediapipe. Since sign languages are sequences of features over the time, this system is hopefully able to utilize the insights gained from landmark detection and propose a robust ASLR system.

2. Scientific Goals

The main goal of this work is to build a system for robust sign language recognition. A landmark-based sign language system contains different aspects and thus raises several problems. A major part of this work is performing Time Series Analysis on the videos of sign language words, and the rest is calibrating the hyperparameters of CNN and RNN models. Specifically:

- Data Collection and Preprocessing
  Gather the videos of the WLASL dataset, where each video sequence represents a word sign.
- Feature Extraction
  Extract hand and pose landmarks from the image frames of each video. The frames are selected using WLASL's proposed fps rate.
- Time Series Representation
  Represent the extracted features of each video as time series data. Each video is a time series of frames, and each frame is a list of landmark features.
- Modeling
  Apply machine learning models to recognize sign language gestures from the time series data. This involve techniques such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNN) applied to the time series data.
- Training and Evaluation
  Train the chosen model on a training set of labeled sign language sequences and evaluate its performance on a separate validation or test set.
- Post-processing
  Post-process the model outputs to improve accuracy and coherence. The refinement involves applying techniques such as sequence alignment (specifically Dynamic Time Wrapping) and data augmentation like sequence jittering, scaling, flipping or rotation.
- Iterative Improvement
  Iterate on the model design, feature extraction methods, and preprocessing techniques based on evaluation results to continuously improve the recognition accuracy and robustness.

3. Scientific Review

Sign language recognition can be categorized based on the physical attributes and intricacy of the gestures. These sign language gestures can be static hand signs, dynamic hand signs characterized by changes in hand shape or orientation, individual gestures involving hand movement (typically representing single words), and continuous sign language, where a series of gestures conveys a distinct meaning, often used for sentences or phrases.

3.1. Static hand posture

Research for static hand posture recognition proposes methods to recognize static hand postures or sign language alphabet by analyzing images of the hands and extract feature vectors based on the static information of the hand shape. Unfortunately, this approach fails to recognize letters of the sign language alphabet which contain local movement made by the wrist, the knuckles, or the finger joints; e.g. the sign for the letter ‘J’ in ASL.

Cui & Swets [6] introduce a self-organizing framework designed to learn and recognize hand signs efficiently. This framework automatically selects the Most Discriminating Features (MDF) using a multi-class, multivariate discriminant analysis, even when working with a limited number of training samples. Their system demonstrates the capability to effectively manage both simple and complex backgrounds. Notably, it achieves a recognition rate of 96% for identifying 28 distinct hand signs, even when these signs were not included in the training phase.

F. Quek [7] propose an inductive learning system capable of deriving rules formulated in disjunctive normal form. The learning algorithm utilizes a feature vector comprising 28 features, including metrics such as the area of the bounding box, hand compactness, and normalized moments. Their results report a recognition rate of 94%.

Triesch & von der Malsburg [8] introduce a view-based system for recognizing hand postures independent of the person in front of a complex background. They utilize an Elastic Graph Matching (EGM) method, previously used for object and face finding and recognition, to represent hand posture features as a graph. They claim a recognition rate of 92.9% for 12 static hand postures signed by 19 individuals against a simple background, and 85.8% against a complex background. Some image frames of their dataset are shown in [Figure 3-1].

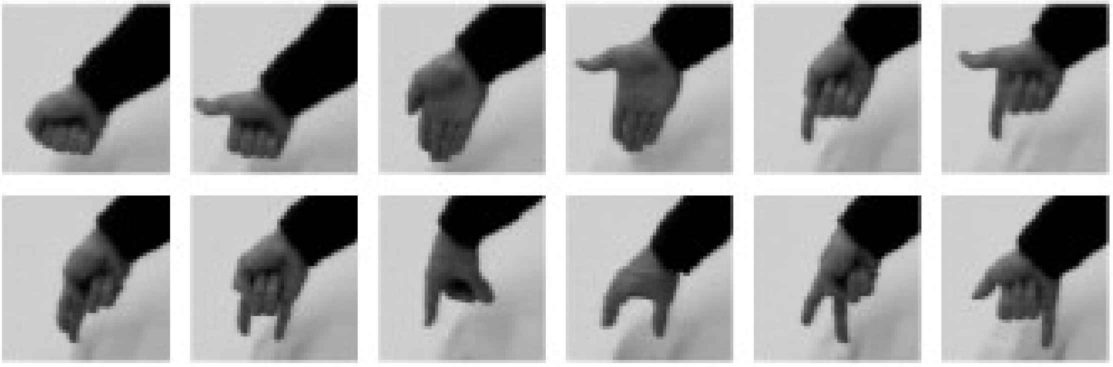


Figure 3-1 Example images from Triesch & von der Malsburg [8] showing the 12 hand postures

Deselaers, Keysers and Ney [9] present a new method for real-time hand pose and object localization and recognition They raise the simultaneous recognition of object classes, hand gestures, and detection of touch events as a unified classification problem. To optimize class discrimination in each image, they utilize a Random Forest algorithm, which dynamically selects and integrates a minimal set of appearance, shape, and stereo features. This streamlined approach not only ensures effective generalization but also enhances efficiency during runtime.

3.2. Dynamic hand posture

The dynamic hand posture and sign language alphabet recognition requires the collection of sequential feature vectors of the gestures. By utilizing dynamic information, they can identify letters that involve local movements. In such cases, local hand movements and changes in hand shape are crucial. These methods focus solely on hand posture changes, disregarding path movements (which involve the shoulder or elbow). Birk et al. [10] designed a system for the recognition of 25 dynamic gestures from the international hand alphabet. As shown in Figure 3-2, the dataset’s image frames capture only the signer’s hand, simplifying hand segmentation. Feature extraction was done with Principal Component Analysis (PCA), yielding an recognition rate of 99% for off-line hand posture recognition.

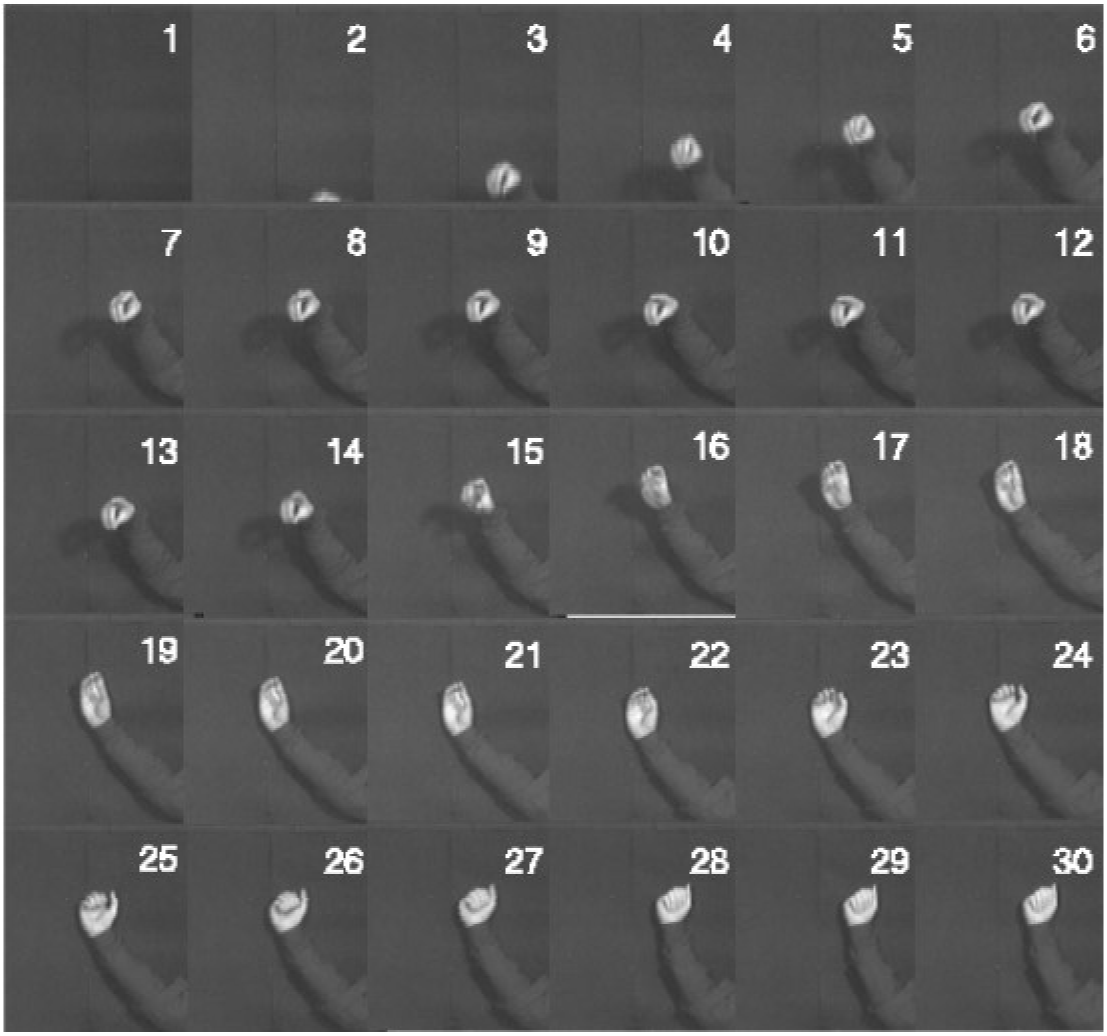


Figure 3-2 Example of sequential image frames in Birk et al. [10 which expresses “P”, “C”, and “A”

Mehdi et al. [11] use a sensor glove for ASL signs feature extraction, and then Artificial Neural Networks (ANN) to recognize 24 letters of the ASL alphabet.

Abe et al. [12] capture the user’s hand movements and hand poses by two cameras in a virtual 3D interface system which allows three-dimensional space maneuvering. The result is a recognition rate of 93.1% for 15 dynamic hand postures.

Malassiottis et al. [13] introduce a system that utilizes a 3D sensor to produce a dense range image of the scene. Unlike systems that rely on color sensors, this approach ensures reliable hand segmentation even under varying lighting conditions.

Hernandez-Rebollar et al. [14] introduce a system capable of recognizing all 26 hand shapes of the ASL alphabet, along with two additional signs: "space" and "enter." This allows users to construct sentences by inputting a sequence of letters and spaces and then submitting them to the system. The system utilizes special data gloves to capture the necessary features. Results show that 21 out of the 26 letters were recognized with 100% accuracy, with the lowest accuracy being 78% for the letter "U."

The Chair of Technical Computer Science at RWTH Aachen (LTI) developed the LTI-Gesture database, which includes 14 dynamic gestures, with 140 training and 140 testing sequences (Akyol et al. [15]). The videos were recorded in a car using an infrared camera, capturing the driver's hand as they performed 14 commands for a media player. The LTI group achieved an error rate of 4.3% on this database (Pelkmann et al. [16]). However, the best error rate of 1.4% was achieved by using appearance-based features and modeling visual variabilities by Dreuw [17] and Dreuw et al. [18, 19].

3.3. Single gesture

In gesture and isolated sign language recognition, both local hand movements and the path movements of the hands are important. As a result, most systems incorporate hand segmentation and tracking.

One such system, described in M.W. Kadous [20] , uses instance-based learning and decision tree learning to recognize 95 isolated Australian sign language words with an 80% recognition rate. The system employs a special data glove called the Power-glove to extract features from the signing. Similarly, a system introduced in H. Matsuo [21] recognizes 38 isolated signs from Japanese sign language, utilizing a pair of stereo cameras to capture 3D movement information.

C. Rigoll et al. [22] introduced a feature extraction method designed to capture dynamic features from sequences of image frames, avoiding the blend of spatial and sequential information, ensuring robustness against variations in gestures and the individuals performing them. An enhanced version of this system was presented in S. Eickeler et al. [23], which is independent of the user's position within the image frames, can reject undefined gestures, and supports continuous online gesture recognition.

Last but not least, in C. Rigoll et al. [22], they introduced a real-time system for recognizing 24 complex dynamic gestures, such as "hand waving," "spin," "pointing," and "head movement." This system is both person and background independent, achieving a recognition rate of 92.9%.

3.4. Continuous sign language

To address the problem of continuous sign language recognition, R. H. Liang et al. [24] developed a system using data gloves that employs a Hidden Markov Model (HMM). This system uses a time-varying parameter threshold of hand posture to identify the end-points of signs within continuous Taiwanese sign language sentences. For a database containing 250 signs, the system achieved an error rate of 19.6%.

C. Vogler and D. Metaxas [25, 26], introduced a framework for recognizing both isolated and continuous ASL sentences using three-dimensional data. This system employs a 3D tracking method with motion-capturing sensors to provide input for a context-dependent HMM classifier. Additionally, the geometric properties of the signer's tracked hand are used to constrain the HMMs. By leveraging 3D features and context-dependent HMMs, the system achieved a recognition rate of 89.91% on a dataset containing 53 ASL signs.

T. Starner et al. [27] developed a real-time system for recognizing continuous ASL sentences using Hidden Markov Models (HMMs). Features are extracted through two methods: a desk-mounted camera and a wearable camera attached to a cap worn by the signer. Figure 3-3 illustrates their video recording setup, as shown in [27]. The system achieved a recognition rate of 92% using the desktop camera and 97% using the wearable camera, on a dataset containing 40 signs.

Bauer & Hienz and Bauer et al. [28, 29] developed a video-based system for recognizing continuous German Sign Language sentences, using a HMM for each sign. The system extracts geometric features from the signers' hands and fingers based on frames captured by a color video camera. To simplify hand and finger segmentation, the signer wears basic colored cotton gloves, as shown in Figure 3-4. The system achieved recognition rates of 94.0% and 91.8% on datasets containing 52 and 97 signs, respectively. By incorporating a language model, these rates improved to 95.4% and 93.2%.

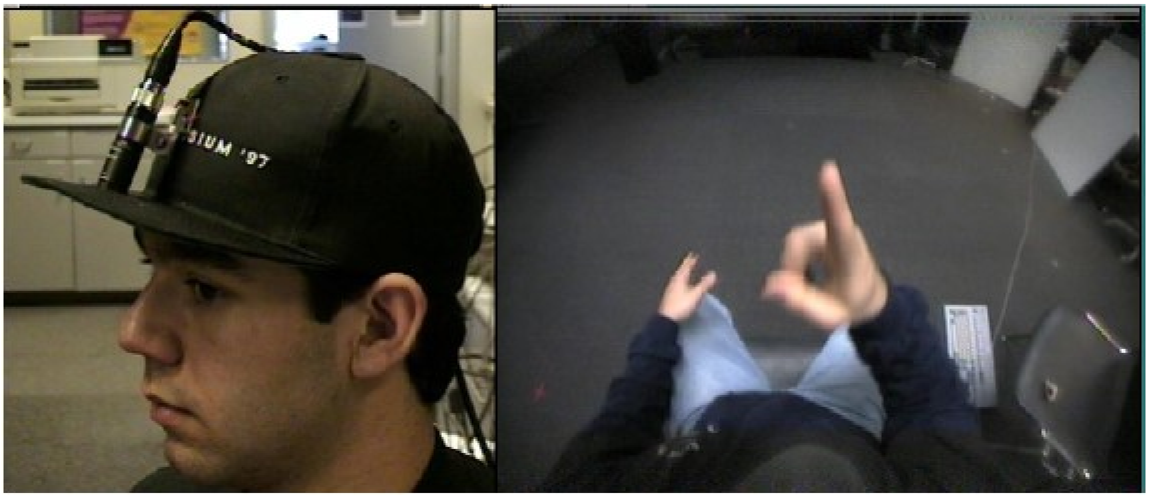


Figure 3-3 A wearable camera which is installed on to a cap worn by the signer is employed as video recording equipment in [Starner et al., 27]

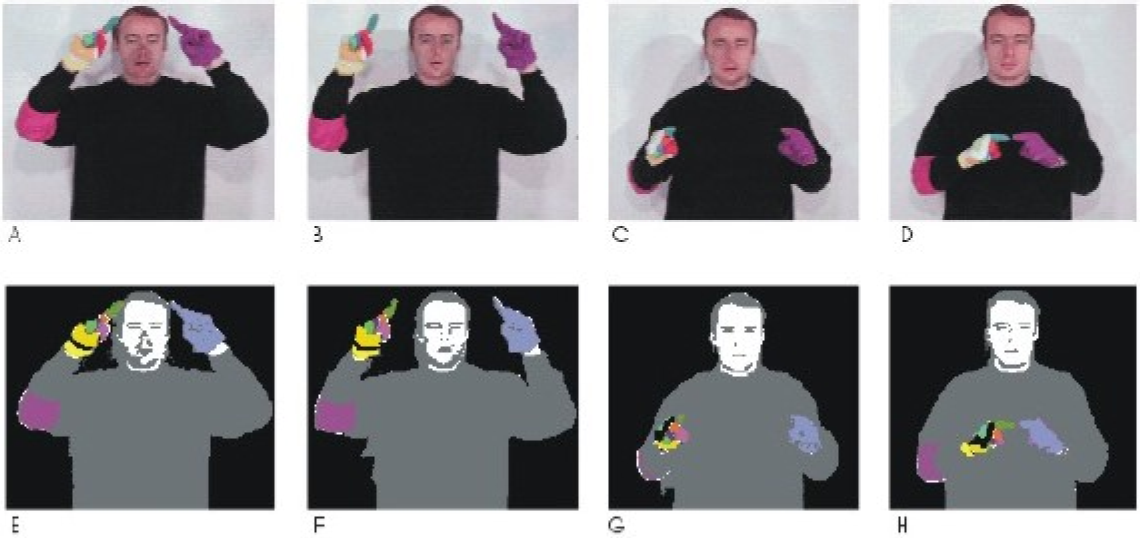


Figure 3-4 Sample frames from colored gloves used in [Bauer et al., 29]. The dominant hand glove is painted with seven different colors to indicate the areas of five fingers, palm and back, and the non-dominant hand glove has another sole color

R. Bowden et al. [30] proposed a two-stage classification process for SLR. In the first stage, a high-level description of hand shape and motion is extracted from the video, focusing on the linguistic aspects of sign language to describe actions at a conceptual level. In the second stage, the system models the temporal transitions of individual signs using a HMM combined with Independent Component Analysis (ICA). The system achieved a recognition rate of 97.67% on a dataset containing 43 signs, with the classifier trained using single-instance samples.

Large vocabulary SLR presents unique challenges, and many researchers have tackled these. For example, Fang et al. [31] propose a fuzzy decision tree combined with heterogeneous classifiers to reduce recognition time without sacrificing accuracy. In large vocabulary recognition, the extensive search space due to numerous classes makes processing time a key concern. Their method, tested on a database of 5,113 signs, reduced recognition time by 11% and improved the recognition rate from 82.9% (achieved by a self-organizing feature maps/Hidden Markov Model (SOFM/HMM) classifier) to 83.7%. Vogler & Metaxas [26] also tackled scalability issues with increasing vocabulary size. Unlike spoken languages, in sign language, phonemes can occur simultaneously, leading to more complex combinations as the vocabulary grows. To address this, they proposed using parallel HMMs, which model these combinations independently during the training process.

Non-manual signals play a crucial role in enhancing the accuracy and completeness of SLR systems, so much effort has been made to recognize the movement and posture of the signer's torso, as well as head movements, facial expressions (such as eye gaze and eyebrow movement), and lip movements. Erdem & Sclaroff [32] developed a system for detecting head movements in ASL videos, which labels relevant head gestures by analyzing the length and frequency of motion signal peaks and valleys. It is specifically designed to identify two key types of periodic head gestures used in sign language communication: "head nods" and "head shakes." H. Kashima et al. [34] introduced an iris detection method able to adapt to various face and eye movements. The system first segments the face region based on color differences from standard skin tone. Next, the eye and mouth regions are extracted using a hybrid template matching technique, and the iris is detected using the saturation and brightness from the eye regions. This system achieves a recognition rate of approximately 98% for eye region detection and 96% for iris detection. Canzler & Dziurzyk [34] analyze lip movements using point distribution models and active shape models, while Vogler & Goldenstein [35] presented a 3D deformable model tracking system capable of recognizing dynamic facial expressions without the need for prior training. For facial feature extraction, it is necessary to track the head of the signer without the interference of things like occlusion of the signer’s hands and the face. Some approaches to head tracking are reported by Cascia et al. [36] and Akyol & Zieren [37].

For all four categories of sign language recognition the use of specialized data acquisition equipment like data gloves, colored gloves, location sensors, or wearable cameras to capture features is a common practice. Although, there is research in the first two categories use simpler stationary cameras without specialized tools, like Triesch & von der Malsburg [8] and Birk et al. [10]. These studies focus on images that capture only the hands, allowing for easy segmentation based on skin color. On the other hand, segmentation becomes more challenging in the last two categories of sign language recognition due to occlusion between the hands and the signer's head, making skin color-based segmentation difficult. To address this, some researchers have explored alternative approaches. For instance, in Starner et al. [27], the camera is positioned above and in front of the signer to reduce occlusion, or in other cases the camera is mounted on a hat worn by the signer, completely avoiding the face. While these methods help to minimize occlusion, they, along with other specialized tools, can be impractical for real-world use.

Automatic SLR has to be practical, as sign language is often the primary means of communication for deaf individuals. Unlike many existing systems, which rely on specialized data acquisition tools, the system proposed in this paper is designed to recognize sign language words and sentences using landmark-based features extracted directly from frames captured by standard cameras. This landmark-based approach offers clear advantages over systems requiring specialized tools, as these tools may already have been used to develop the landmark recognition models. More importantly, it enables real-world applications where the use of special equipment is not possible. A standard laptop with a fixed-position camera could provide real-time sign language translation in stores, offices, and other everyday places.

4. Datasets

All automatic SLR systems, particularly statistical ones, depend on sufficiently large corpora. A appropriate corpus of SL data is required for the training. To store and process sign language data, a textual representation of the signs is necessary. Although various notation systems exist to capture different linguistic aspects, we focus on gloss notation. Glosses are widely used to transcribe sign language video sequences and serve as a semantic representation of sign language, making them essential for training and analyzing recognition systems.

In this project, a gloss is a word describing the content of a sign. Manually annotating sign language videos is challenging, often leading to inconsistencies in notation within a single corpus.

4.1. Public word-level ASL datasets

There are four publicly released word-level ASL datasets: WLASL [38], Boston ASLLVD [39], Purdue RVL-SLLL ASL Database [40] and RWTH-BOSTON-50 [41].

The Purdue RVL-SLLL ASL Database [40] includes 39 motion primitives with various hand shapes commonly used in ASL. Each of these primitives is performed by 14 native signers. It is worth mentioning that the primitives represent components of ASL signs but may not correspond directly to individual English words.

The Boston ASLLVD [39] dataset contains 2,742 glosses (i.e., words) with 9,794 total examples. While it covers a broad vocabulary, over 2,000 of these glosses have three or fewer examples, making it less ideal for training large-scale classifiers with thousands of categories.

The RWTH-BOSTON-50 [41] dataset features 483 samples of 50 different glosses signed by 2 signers. The RWTH-BOSTON-104 dataset includes 200 continuous sentences performed by 3 signers, covering 104 signs. The RWTH-BOSTON-400, a sentence-level corpus, consists of 843 sentences containing approximately 400 signs, performed by 5 signers.

While all the previously mentioned datasets offer valuable contributions and distinct approaches to addressing the word-level sign recognition task, they fall short in fully capturing the complexities of the task. This is primarily due to the insufficient number of instances and the limited diversity of signers, which are critical for building robust and generalizable recognition systems.

Existing word-level SLR models are predominantly trained and evaluated on either private datasets [42, 43, 44, 45, 46] or small-scale datasets with fewer than 100 words [43, 44, 45, 46, 47, 48, 49]. These models typically consists of three stages: feature extraction, temporal-dependency modeling, and classification. Initially, various hand-crafted features are used to represent static hand poses, such as SIFT-based features [50, 51, 52], HOG-based features [53, 54, 55], and features in the frequency domain [56, 57]. To capture the temporal relationships in video sequences, HMM [58, 59] are used. Additionally, Dynamic Time Warping (DTW) [60] is used to address variations in sequence lengths and frame rates. Finally, classification algorithms like Support Vector Machines (SVM) [61] are applied to label the signs with their corresponding words.

Following the action recognition methodology, some systems [62, 63] have employed Convolutional Neural Networks (CNNs) to extract holistic features from image frames, using these features for classification. Other approaches [64, 65] first extract body landmarks and then concatenate their locations into a feature vector, which is fed into a stacked Gated Recurrent Unit (GRU) to recognize signs. These methods highlight the effectiveness of leveraging human poses for word-level sign recognition tasks. Instead of handling spatial and temporal information separately, some recent methods use 3D CNNs [45, 66] to capture spatial-temporal features at the same time. However, these approaches have only been tested on small-scale datasets, leaving their generalization ability uncertain. Additionally, the absence of a standardized large-scale word-level sign language dataset makes it difficult to compare results across methods evaluated on different small datasets, which raises concerns about their practical applicability in real-world scenarios.

To solve the previous problems in SLR, WLASL introduces a large-scale word-level American Sign Language (ASL) dataset, known as the WLASL database. Since this dataset contains only RGB videos, algorithms trained on it can be seamlessly applied to real-world situations with minimal equipment requirements, such as standard cameras. Furthermore, WLASL provides a set of baseline models using state-of-the-art sign recognition methods, which helps facilitate the evaluation and comparison of future SLR research efforts.

4.2. WLASL

To create a large-scale, signer-independent ASL dataset, WLASL sourced videos from two primary internet sources. The first source was educational sign language websites, such as ASLU [67] and ASL-LEX [68], which offer accurate mappings between ASL glosses and signs, as the videos are reviewed by experts before being uploaded. The second source was ASL tutorial videos from YouTube, where WLASL selected videos with clear titles that describe the gloss of the sign being demonstrated. In total, the dataset compiled 68,129 videos covering 20,863 ASL glosses from 20 different websites. Each video features a signer performing a single sign (with single or multiple repetitions) in a nearly frontal view, but with varying backgrounds.

After gathering the video resources, WLASL applied a filtering process to refine the dataset. Videos with gloss annotations consisting of more than two English words were removed to ensure the dataset focuses on individual words only. Additionally, glosses with fewer than seven video samples were excluded to ensure that there were sufficient examples for splitting into training and testing sets. Glosses with limited video samples likely represent less commonly used words, so their removal does not significantly impact the practical usefulness of the dataset. After this selection process, the dataset was reduced to 34,404 video samples covering 3,126 glosses, ready for further annotation.

In addition to providing a gloss label for each video, the WLASL dataset includes various types of metadata, such as temporal boundaries, body bounding boxes, signer annotations, and sign dialect/variation annotations.

Temporal boundary. Temporal boundaries indicate the start and end frames of each sign. For videos without sign repetitions, the boundaries are marked at the first and last frames of the sign. In cases where videos contain repeated signs, boundaries are labeled between repetitions. Only one instance of a repeated sign is kept to ensure that the same signer performing the same sign does not appear in both the training and testing sets. This prevents models from overfitting to the test set and improves generalization.

Body Bounding-box. To minimize the impact of background and ensure that models focus on the signers, WLASL employed YOLOv3 [69] as a person detection tool to identify the body bounding boxes of signers in the videos. Since the size of the bounding box can vary as a person performs signs, WLASL used the largest bounding box size throughout the video to crop the signer consistently. This approach ensures that the signer remains the focus of the video, regardless of variations in their movements during signing.

Signer Diversity. A robust sign recognition model needs to handle variations between different signers, such as differences in appearance and signing pace, in order to generalize well in real-world scenarios. For example, as illustrated in Figure 4-1, two signers may perform the same sign with slightly different hand positions. To ensure that the dataset captures such diversity, WLASL identified the signers in its dataset and provided signer IDs as part of the video's metadata. To achieve this, WLASL used a face detector and FaceNet's [70] face embedding technique to encode the faces of the signers in the dataset. The Euclidean distances between the face embeddings were then compared. If the distance between two embeddings was below a predefined threshold (0.9), the system considered those videos to be signed by the same person. After the initial automatic labeling, a manual review was conducted to correct any mislabeling, ensuring accurate identification of the signers.

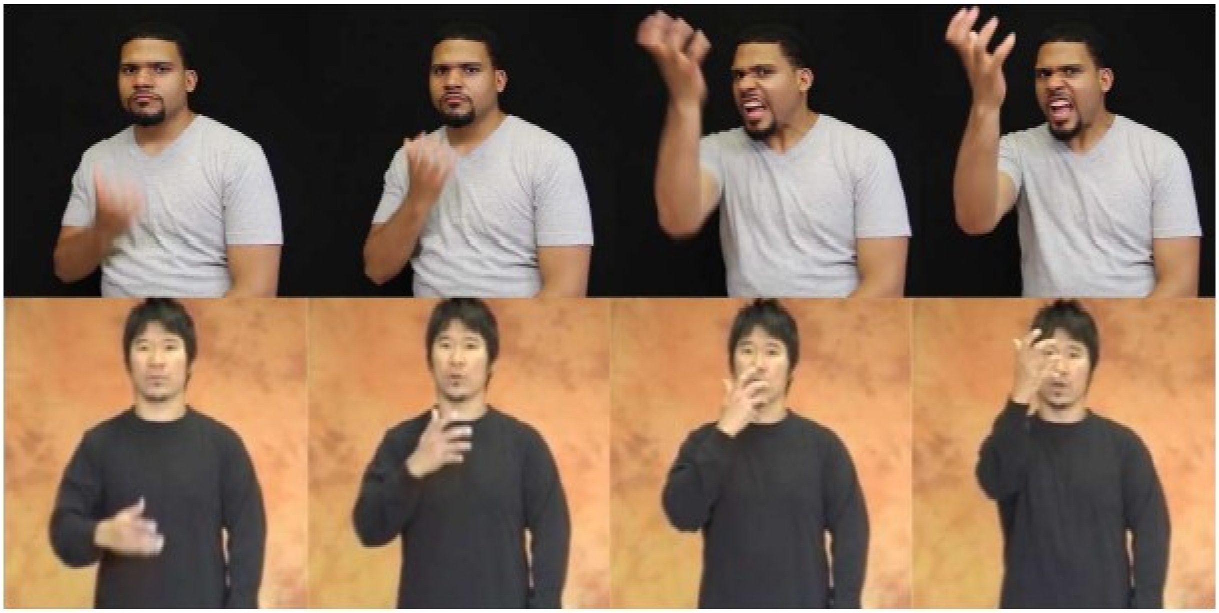


Figure 4-1 Signers perform “Scream” with different hand positions and amplitude of hand movements.

Dialect Variation Annotation. Similar to spoken languages, ASL also has dialect variations [71], which may involve differences in hand shapes and motions. To ensure that dialect variations do not only appear in the testing dataset, WLASL manually labels these variations for each gloss. To achieve this, annotators receive training beforehand to understand the basics of ASL and to accurately differentiate between signer-specific variations and dialect variations. In order to make the annotation process faster and maintain quality, WLASL developed an interface where annotators compare signs from two videos shown side by side. This system automatically counted the number of dialects and assigned labels to each variation. After completing the dialect annotations, each video was given a corresponding dialect label. This process ensures that dialect-specific signs in the testing set have matching examples in the training set. Additionally, any sign variation with fewer than five examples was discarded, as these would not provide sufficient samples to split into training, validation, and testing sets.

5. Methodology

This project aims to perform time series analysis on video sequences of isolated sign language words, and building a robust and accurate ML model for Sign Language Recognition of isolated sign language word gestures. The next chapters describe in detail the decisions and methodology followed to achieve this.

5.1. Data Collection and Preprocessing

WSAL, as described, proposes a method of creating the dataset locally. The proposed way, as described in [38], creates a balanced dataset. The source of the videos is the internet, and thus there is no guarantee that the links will be always available. WLASL has a total of 68,129 videos of 20,863 ASL glosses from 20 different websites that can be potentially downloaded. Li et al. fir their work [38] ended up with 34,404 video samples of 3,126 glosses, after the essential selection to keep the standards of the dataset. The time of this work 13,632 were available for 2,000 glosses.

The pool for our experiments is a corpus of 20, one of 100 and one of 300 glosses. Each corpus contains different words (glosses) from another. During the selection of the glosses, an effort was made to get the ones with as many video instances as possible. For the largest corpus size of 300, the minimum number of video instances was 7, which is WLASL's minimum to safeguard the datasets balance. For the 20 and 100 ones the minmum number of instances was higher.

The labels of each corpus were encoded using One-hot Encoding. One-hot encoding is a process used to represent categorical variables as binary vectors. It was used as the labels in SLR are categorical, and categorical variables need to be converted into a numerical format that can be used as input for training.

5.2. Features extraction and Time Series representation

In sign language the message is expressed through movement of the hands, arms, shoulders and face. To extract a complete and descriptive set of features we will target the landmarks of the hands, and the upper body (arms and shoulders). An argument can be made that the face also participates in facial expressions. This project did not follow this direction, as the main focus of this work and isolated signing are the hands and arms, in contrast to continuous sign language recognition where facial expressions play a far more impactful role.

To extract the desired features, Google Mediapipe's Hand Landmarks Detection and Pose Landmarks Detection were used.

Hand Landmarks Detection. The Hand Landmarker task lets you detect the landmarks of the hands in an image. This task is used to locate key points of hands and render visual effects on them. It operates on image data with a ML model as static data or a continuous stream and outputs hand landmarks in image coordinates, hand landmarks in world coordinates and handedness(left/right hand) of multiple detected hands.

The hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions. The model was trained on approximately 30K real-world images, as well as several rendered synthetic hand models imposed over various backgrounds.

The hand landmarker model bundle contains a palm detection model and a hand landmarks detection model. The Palm detection model locates hands within the input image, and the hand landmarks detection model identifies specific hand landmarks on the cropped hand image defined by the palm detection model.

Since running the palm detection model is time consuming, when in video or live stream running mode, Hand Landmarker uses the bounding box defined by the hand landmarks model in one frame to localize the region of hands for subsequent frames. Hand Landmarker only re-triggers the palm detection model if the hand landmarks model no longer identifies the presence of hands or fails to track the hands within the frame. This reduces the number of times Hand Landmarker tiggers the palm detection model.

In Figure 5-1 all the detected landmarks are shown.

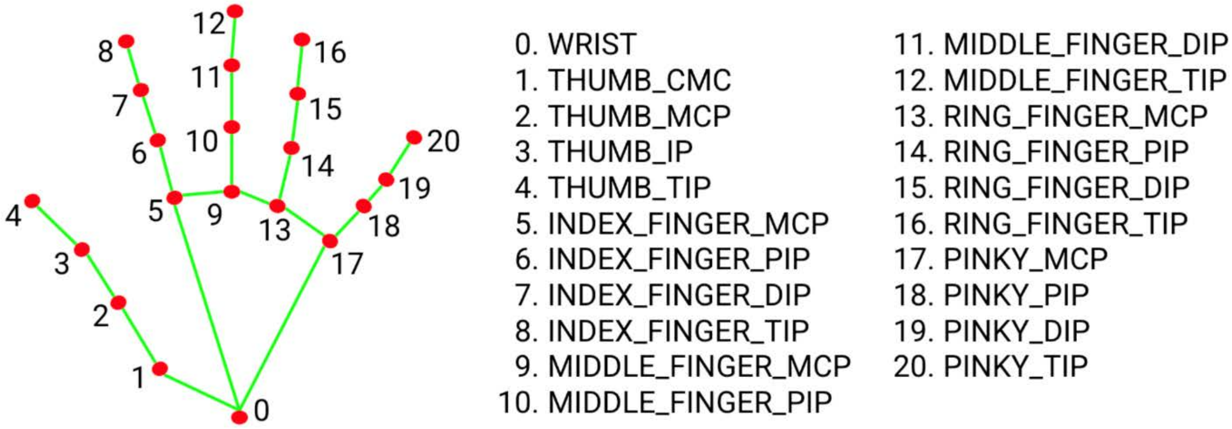


Figure 5-1 The detected hand landmarks by Mediapipe

The result of the detection contains the following basic info:

- Handedness

  Handedness represents whether the detected hands are left or right hands.

- Landmarks

  There are 21 hand landmarks, each composed of x, y and z coordinates. The x and y coordinates are normalized to [0.0, 1.0] by the image width and height, respectively. The z coordinate represents the landmark depth, with the depth at the wrist being the origin. The smaller the value, the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

- World Landmarks

  The 21 hand landmarks are also presented in world coordinates. Each landmark is composed of x, y, and z, representing real-world 3D coordinates in meters with the origin at the hand's geometric center.

This works utilizes the Handiness and the normalized Landmarks, since the WLASL dataset has videos with various dimentions. Figure 5-2 shows the result of the Hand Landmarks Detection task on an image.

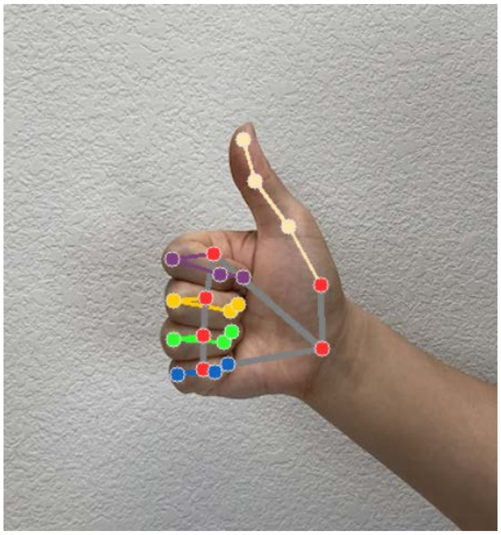


Figure 5-2 Example of landmark detection on images

Pose Landmarks Detection. The MediaPipe Pose Landmarker task lets you detect landmarks of human bodies in an image or video. This task can be used to identify key body locations, analyze posture, and categorize movements. This task uses ML models that work with single images or video. The task outputs body pose landmarks in image coordinates and in 3-dimensional world coordinates.

The Pose Landmarker uses a series of models to predict pose landmarks. The first model detects the presence of human bodies within an image frame, and the second model locates landmarks on the bodies.

The following models are packaged together into a downloadable model bundle:

- Pose detection model: detects the presence of bodies with a few key pose landmarks.
- Pose landmarker model: adds a complete mapping of the pose. The model outputs an estimate of 33 3-dimensional pose landmarks.

This bundle uses a convolutional neural network similar to MobileNetV2 and is optimized for on-device, real-time fitness applications. This variant of the BlazePose model uses GHUM, a 3D human shape modeling pipeline, to estimate the full 3D body pose of an individual in images or videos.

The pose landmarker model tracks 33 body landmark locations, representing the approximate location of the body parts shown in Figure 5-3.

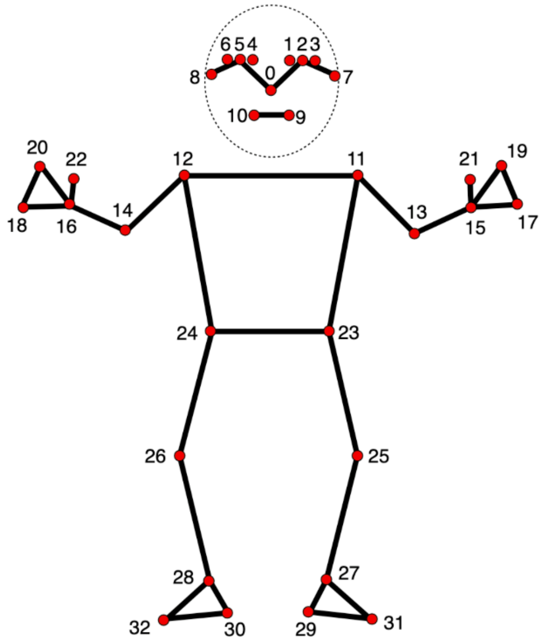


Figure 5-3 Mediapipe pose landmarks

From theses landmarks a subset was selected, as shown in Figure 5-4.

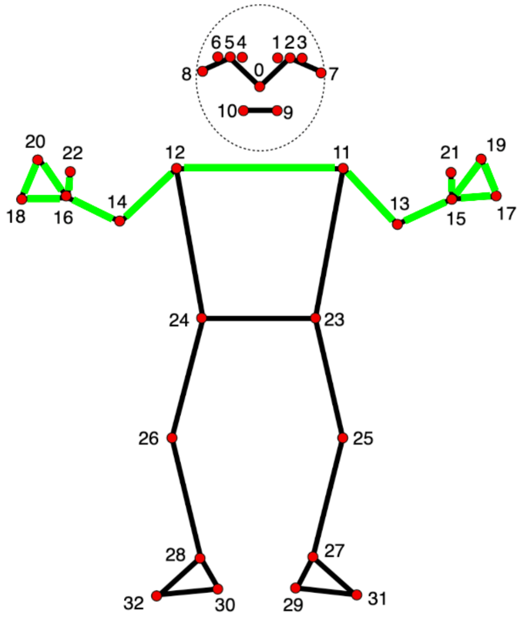


Figure 5-4 Selected pose landmarks for this project

The model output contains both normalized coordinates (Landmarks) and world coordinates (WorldLandmarks) for each landmark. Once again, the normalized Landmarks were selected.

Initially, using OpenCV every video instance was parsed. While parsing the videos, the number of maximum detected hands was set to two. The frame rate, starting frame and ending frame were set according to WLASL's specification. Then the video was read frame by frame. On every frame the Landmarkers extracted the desired landmarks, which we saved as lists in text files. The final output was a text file where every line is a list. Every such list is a 2D list, and contains as many list as the frames of that video. Every such sublist has the following structure:

```
        [left_hand_landmarks, right_hand_landmarks, pose_landmarks]
```

where

```
        left_hand_landmarks = [0.0]*3*21

        right_hand_landmarks = [0.0]*3*21

        pose_landmarks = [0.0]*3*12
```

That is because every hand consists of 21 landmarks, and every landmark of 3 values x, y, and z with x, y, z ∈ $[0, 1]$

Respectively, 12 pose landmarks were chosen for our experiments with 3 values x, y, z each and x, y, z ∈ $[0, 1]$

6. Modeling

6.1. Proposed model architectures

For SLR some of the most common and efficient models that are used are Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), combined models like Convolutional Recurrent Neural Networks (CRNNs), or hybrid architectures. In this work CNNs and RNNs were utilized either on their own or as a hybrid.

CNN model. CNNs are widely used for image-based recognition tasks, making them suitable for SLR from video frames. They excel at learning spatial patterns and hierarchies of features, which are crucial for recognizing hand gestures.

CNN is designed for optimal performance by a proper selection of convolution layers and the number of neurons. The selection of the number of neurons and convolution layers vary depending the task, the dataset, and the pre-processing that was performed. The proposed CNN architecture maximizes the recognition accuracy. It consists of one masking layer, six convolution layers, three pooling layers, a dropout layer, and a fully connected (dense) layer. The steps of the proposed CNN model are described in the Figure 6-1.

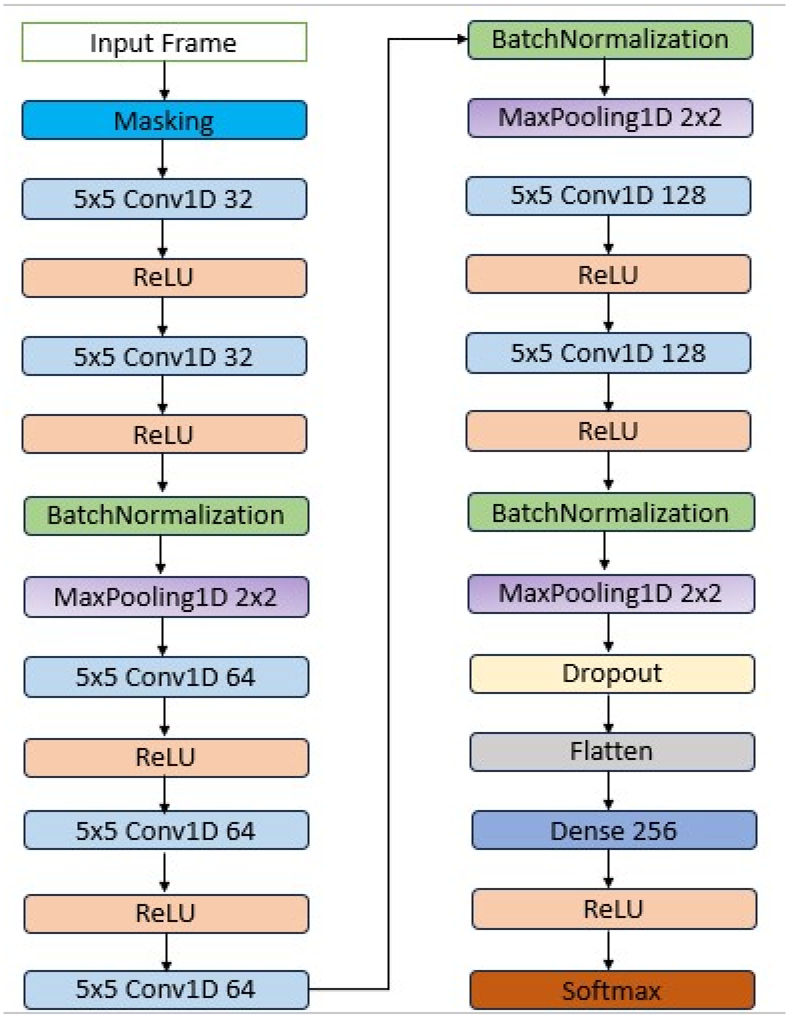


Figure 6-1 CNN model architecture

RNN model. RNNs are suitable for sequential data processing, making them valuable for capturing temporal dependencies in sign language sequences. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are common RNN variants used in SLR. RNNs are effective for modeling the temporal dynamics of sign language gestures over time.

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem encountered in traditional RNNs. LSTM networks are particularly well-suited for sequence prediction and time-series analysis tasks due to their ability to capture long-term dependencies in data.

LSTMs utilize three types of gates to regulate the flow of information within the network:

- Forget Gate: Decides what information to discard from the cell state.
- Input Gate: Determines what new information to store in the cell state.
- Output Gate: Controls the information to be output based on the current input and the previous state.

LSTMs are trained using Back-Propagation Through Time (BPTT), a variant of Back-Propagation suitable for sequence data. Gradient descent algorithms are used to update the weights of the network based on the error between predicted and actual outputs. The steps of the proposed LSTM model are described in the Figure 6-2.

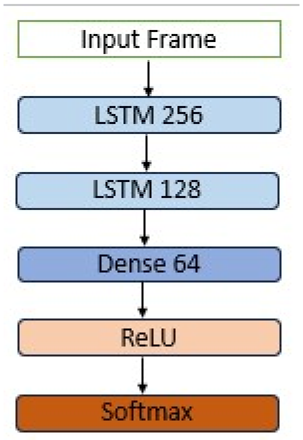


Figure 6-2 LSTM model architecture

CNN – LSTM Hybrid model. A hybrid CNN-LSTM model combines the strengths of CNNs and LSTM networks to effectively capture spatial and temporal features in sequential data. This architecture is particularly well-suited for tasks involving sequential data with both spatial and temporal dependencies, such as video analysis, action recognition, and time-series forecasting.

The output of the CNN layers is fed into the LSTM layers, which sequentially process the spatial features extracted from the input data. The LSTM layers capture the temporal relationships between the spatial features across different time steps, allowing the model to learn long-term dependencies and temporal dynamics. The steps of the proposed LSTM model are described in the Figure 6-2.

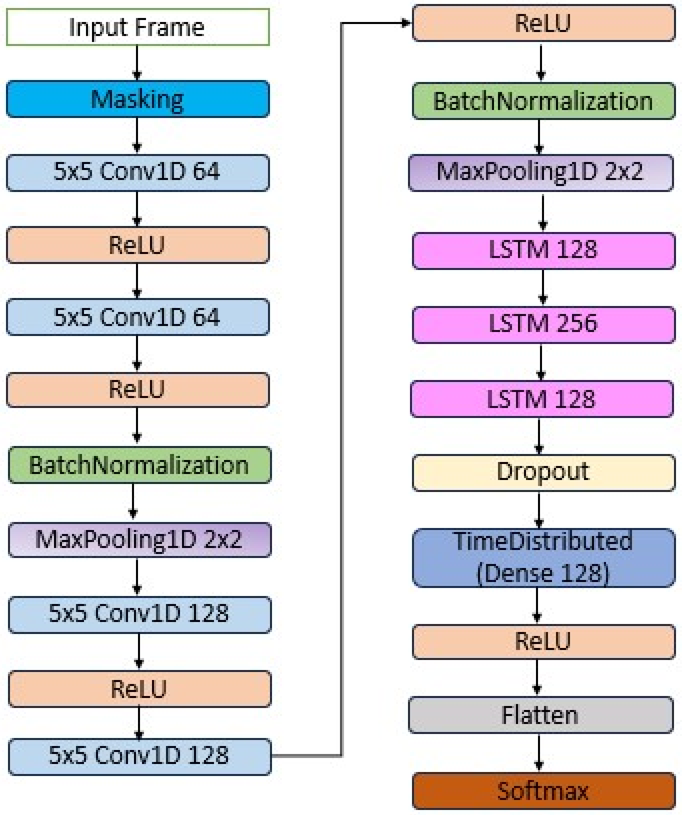


Figure 6-3 CNN-LSTM hybrid model architecture

6.2. Model hyperparameters

As far as the configuration of the basic NN layers go (number of neurons, filters, or kennels), the decision was made based on experience and experimentation. However, for the following hyperparameters the logic behind the decision is as follows:

Activation Function. In a CNN architecture, the activation function decides which node should be fired at a time. The ReLU activation function was chosen as it substitutes all negative values to 0 and remains identical with the positive values. The selection of ReLU is a common choice in SLR models both for its speed and effectiveness. ReLU tends to be accelerate the training process, in comparison to other functions (e.g sigmoid), and it can diminish the problem of gradient vanishing. The ReLU function is expressed by the following formula (1) :

$$\mathrm{ReLU(y) = \ max(0, y), \ where\ y\ refers\ to\ the\ input\ to\ a\ neuron} \qquad (1)$$

Pooling Layer. Pooling also speeds up the training process, but more importantly it reduces the memory requirements of the network by reducing the links between the convolutional layers. Specifically, the Max Pooling, which was used, downsamples feature maps, reducing their spatial dimensions while retaining important information. Here a pool size of 2 was selected. The output dimension of the max-pooling operation can be calculated by the following formula:

$$N_{out} = floor\left(\frac{N_{in}-F}{S}\right) + 1 \qquad (2)$$

where N_{in}, F and S refer to the size of the input image, kernel, and stride, respectively.

Dropout. Dropout is a form of regularization that aims to reduce overfitting by preventing complex co-adaptations of neurons. It does this by randomly setting a fraction of input units to zero during each update of the network parameters. The selected dropout in this study is 0.2.

Output Layer. The output layer of a neural network is the final layer that produces the predictions or outputs of the model. Its structure and activation function depend on the type of task the neural network is designed for. For multi-class classification tasks with N classes, the output layer typically consists of N neurons, each representing the probability of belonging to a specific class. The activation function is usually softmax, which ensures that the probabilities sum up to 1 across all classes. The mathematical formula for the Softmax function is given by:

$$\sigma(X)_i = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \qquad \textit{for i = 1, 2, 3, …K} \qquad (3)$$

where, x_i are the inputs from the previous layer used to each Softmax layer node and K is the number of classes.

Batch normalization. Batch normalization is a technique used to normalize the inputs of each layer in a neural network, aiming to stabilize and accelerate the training process. It was introduced by Sergey Ioffe and Christian Szegedy in [72].

Flatten. The Flatten layer is a Tensorflow Keras class that is used to flatten the input into a one-dimensional array. It is typically used to transition from a multidimensional input, such as the output of convolutional layers in a CNN, to a fully connected (dense) layer in a neural network.

6.3. Dynamic Time Wrapping and Data Augmentation

Dynamic Time Warping (DTW) is a technique used to align and compare sequences of varying lengths in a robust and flexible manner. It is commonly applied in fields such as speech recognition, gesture recognition, and time-series analysis, where the temporal structure of the data may vary.

For the purpose of this work, the first video of every gloss of the corpus is used as reference. The rest of the video instances of that gloss are aligned based on that reference. For the implementation of DTW the python library FastDTW was used. FastDTW is an approximation algorithm for DTW that provides a computationally efficient solution for aligning and comparing sequences with varying lengths. FastDTW aims to significantly reduce the computational complexity of DTW while still producing reasonably accurate alignments.

The DTW was applied first on the raw extracted landmark sequences. Next, four different data augmentations techniques were applied on the sequences.

Data augmentation is a technique used to artificially increase the size of the dataset by applying various transformations to the existing data samples. Data augmentation helps improve the generalization ability of models by exposing them to a wider range of variations in the input data, thereby reducing overfitting and improving performance on unseen data. The transformations that were applied are the following:

- *Spatial jittering*, also known as spatial augmentation or geometric transformation, is a technique commonly used to introduce variations in the spatial structure of images (landmark sequences in this case). It involves applying geometric transformations to the original images to create new, slightly modified versions.

- *Rotation* is a common data augmentation technique particularly when working with image datasets. It involves rotating the original images by a certain angle to introduce variations and enhance the model's ability to generalize.
- *Scaling* is a technique that helps introduce variations in the scale or size of images. It involves resizing the original images to different dimensions, either larger or smaller, to augment the dataset and improve the model's ability to generalize.
- *Flipping* is used to introduce variations in the orientation or direction of images. It involves flipping the original images horizontally, vertically, or both to augment the dataset and improve the model's ability to generalize.

The previous techniques refer to image datasets, but the same principles apply for landmark sequences extracted from video frames.

7. Experimental Results

In this chapter the results of our experiments are presented. As mentioned before, this work uses as it's the WLASL dataset. From this dataset we produce three corpuses of 20, 100 and 300 glosses. On these corpuses we will train the proposed ML models. Each model will be trained on the dataset with DTW applied and without. The data that are fed to the model are randomly split into training and test data with ratio of 0.2.

In every training Early Stopping has been applied, monitoring the validation loss. Early stopping is a technique used to prevent overfitting and improve the generalization ability of models. It works by monitoring the performance of the model on a validation dataset during training and stopping the training process when the performance stops improving or starts deteriorating. This is done to prevent the model from continuing to learn the training data too well and memorizing noise or outliers, which can lead to poor performance on unseen data.

In the following three sub-chapters, we present the results of the training for each corpus. In the final chapter, we compare those results to conclude how the corpus size affects the efficiency of the proposed models.

7.1. Corpus 20

This corpus consists of the first 20 glosses of the WLASL dataset. On average every gloss has 15 video instances assigned to it. The following tables present the progression of each model during epochs.

Table 1 CNN (corpus 20) model progression for the first 40 epochs

CNN – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.1642	2.9850	0.1551	2.8203
2	0.3969	1.8989	0.1089	2.7394
3	0.6493	1.0770	0.1155	3.1821
5	0.9315	0.2276	0.1353	3.6691
8	0.9926	0.0371	0.2937	2.3827
10	0.9893	0.0531	0.7129	1.0299
20	0.9439	0.2154	0.8020	0.6121
30	0.9975	0.0165	0.9868	0.0276
40	1.0000	0.0039	1.0000	0.0039

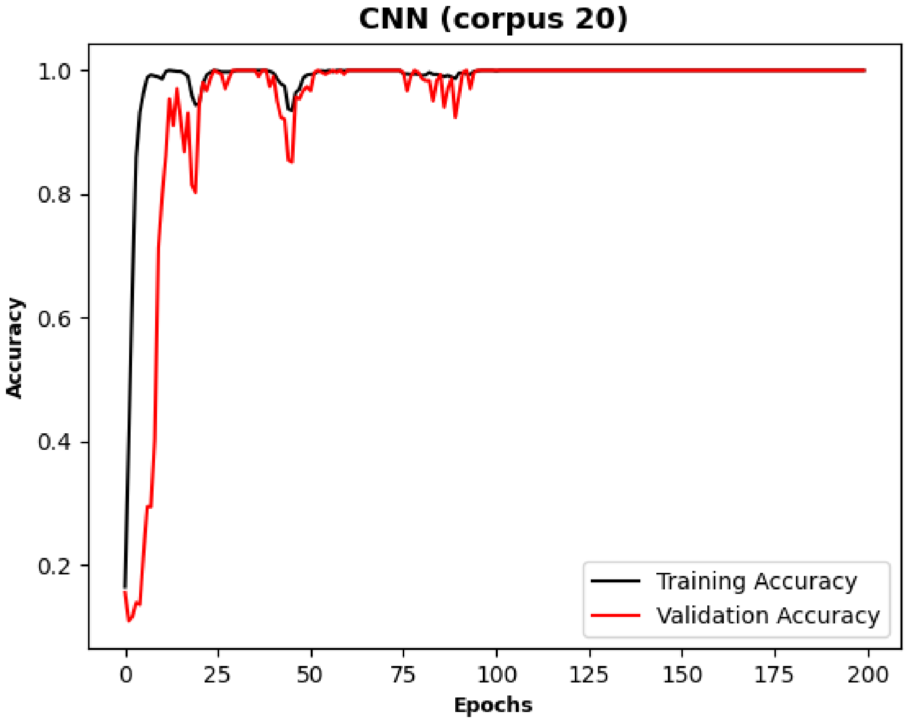


Figure 7-1 CNN (corpus 20) training & validation accuracy through epochs

The CNN reached high accuracy (93.15%) by the fifth epoch, but the validation accuracy was too low (13.53%). This indicates heavy overfitting. However, from there the model got significantly better every 10 epochs, until epoch 30 where the model achieved 99% accuracy on both training and validation accuracy. At later epochs there is some small drops, but they still remain above 90%.

Table 2 CNN with DTW (corpus 20) model progression for the first 40 epochs

CNN with DTW – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.3787	2.2304	0.3069	2.6703
2	0.7104	0.8889	0.2805	2.3561
3	0.8828	0.3656	0.3894	2.1107
5	0.9538	0.1533	0.5314	1.6431
8	0.9884	0.0419	0.7360	0.8831
10	0.9942	0.0249	0.9769	0.1288
20	0.9802	0.0540	0.9109	0.3413
30	0.9901	0.0248	0.9538	0.1587
40	0.9959	0.0169	0.9934	0.0111

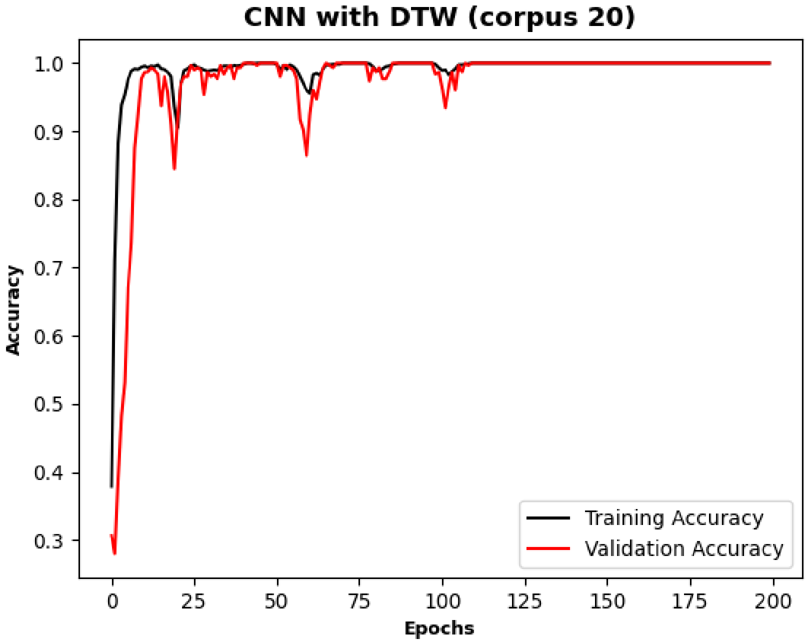


Figure 7-2CNN with DTW (corpus 20) training & validation accuracy through epochs

DTW helped the CNN achieve three times faster high training and validation accuracy. By epoch 10, the CNN has hit the 95% threshold; 97% to be exact.

On the other hand, the LSTM architecture for the given time frame of 200 epochs did not perform so well. The best it achieved was a 60% accuracy. Even when left to train for more epochs (up to 500), the results did not improve. The exact picture of the model is shown at Table 3. In Figure 7-3, it is evident that the validation accuracy is about 5% lower than the training accuracy. Even though this does not indicate overfitting, it shows that even this low accuracy the model will perform even worse than expected.

The DTW did not help the final accuracy of the model, as shown in Figure 7-4. It helped the model to have consistently almost equal validation accuracy. That means that the LSTM model did not learn to recognize the 20 words well, but the 60% accuracy will actually recognize 6 out of 10 word-signs correctly.

Table 3 LSTM (corpus 20) model progression for the first 200 epochs

LSTM – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0512	2.9930	0.0858	2.9741
10	0.2896	2.1966	0.2838	2.1520
20	0.3861	1.8878	0.3630	1.8716
30	0.4183	1.7576	0.4422	1.7856
40	0.4381	1.6614	0.4719	1.6094
50	0.4158	1.6593	0.4851	1.5404
60	0.4686	1.4868	0.4554	1.5142
70	0.4563	1.5200	0.4851	1.5937
80	0.4868	1.4333	0.4653	1.5069
90	0.4827	1.3828	0.5380	1.3378
100	0.4967	1.3369	0.5413	1.2997
110	0.5116	1.3072	0.5248	1.3369
120	0.5512	1.1952	0.5380	1.3129
130	0.5602	1.1867	0.5479	1.2528
140	0.5627	1.1596	0.4851	1.4833
150	0.5701	1.1656	0.5182	1.3138
160	0.5932	1.0744	0.5941	1.1081
170	0.5941	1.0362	0.6172	1.0774
180	0.5990	1.0316	0.5545	1.2169
190	0.4686	1.5142	0.3795	1.8888
200	0.6469	0.9444	0.6139	1.0243

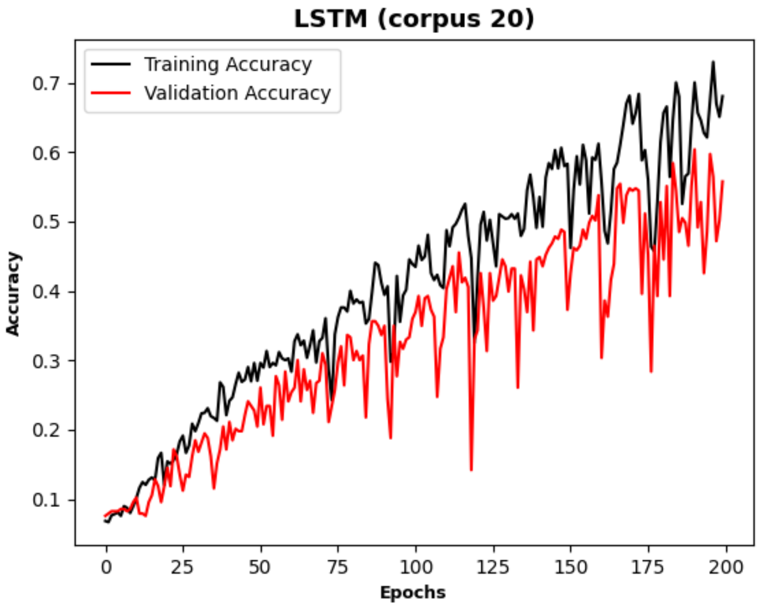


Figure 7-3 LSTM (corpus 20) training & validation accuracy through epochs

Table 4 LSTM with DTW (corpus 20) model progression for the first 200 epochs

LSTM with DTW – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0512	2.9930	0.0858	2.9741
10	0.2896	2.1966	0.2838	2.1520
20	0.3861	1.8878	0.3630	1.8716
30	0.4183	1.7576	0.4422	1.7856
40	0.4381	1.6614	0.4719	1.6094
50	0.4158	1.6593	0.4851	1.5404
60	0.4686	1.4868	0.4554	1.5142
70	0.4563	1.5200	0.4851	1.5937
80	0.4868	1.4333	0.4653	1.5069
90	0.4827	1.3828	0.5380	1.3378
100	0.4967	1.3369	0.5413	1.2997
110	0.5116	1.3072	0.5248	1.3369

120	0.5512	1.1952	0.5380	1.3129
130	0.5602	1.1867	0.5479	1.2528
140	0.5627	1.1596	0.4851	1.4833
150	0.5701	1.1656	0.5182	1.3138
160	0.5932	1.0744	0.5941	1.1081
170	0.5941	1.0362	0.6172	1.0774
180	0.5990	1.0316	0.5545	1.2169
190	0.4686	1.5142	0.3795	1.8888
200	0.6469	0.9444	0.6139	1.0243

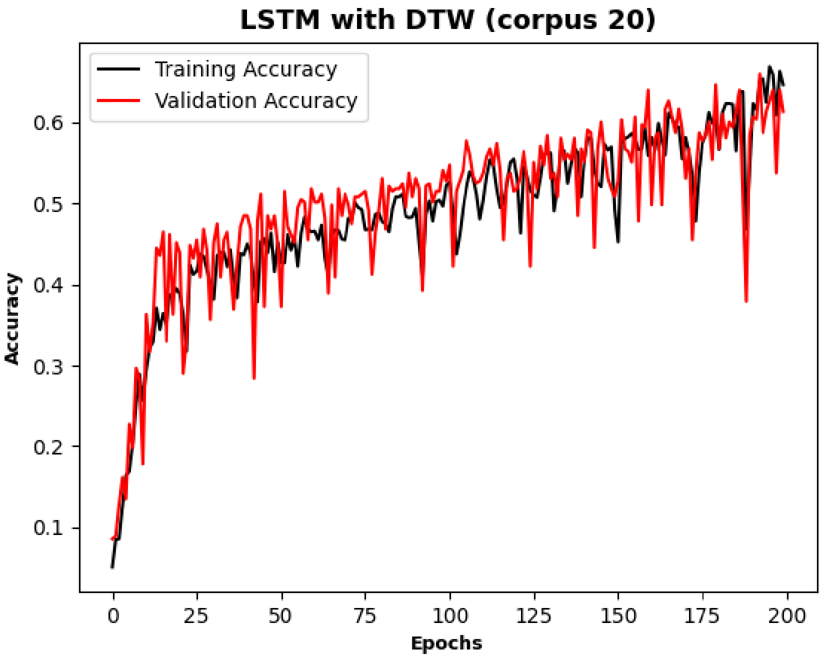


Figure 7-4 LSTM with DTW (corpus 20) training & validation accuracy through epochs

The hybrid CNN-LSTM model had an equally good accuracy with the CNN model, if not slightly better. The DTW did not affect the model’s efficiency. In particular, the were more spikes and drops compared to CNN or to the hybrid without DTW, as shown in Figure 7-6.

Table 5 CNN-LSTM hybrid (corpus 20) model progression for the first 60 epochs

CNN LSTM Hybrid – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.1304	2.6965	0.1881	2.6802
2	0.1774	2.4264	0.1716	2.4525
3	0.2690	2.1858	0.2343	2.3837
5	0.3630	1.8250	0.2376	2.0783
8	0.5223	1.3445	0.4059	1.5990
10	0.6980	0.9177	0.5380	1.3368
15	0.8870	0.3305	0.8878	0.3566
20	0.9513	0.1580	0.8515	0.5254
30	0.9744	0.0758	0.9769	0.1059
40	0.9983	0.0072	0.9967	0.0116
50	0.9926	0.0240	0.9736	0.0851
60	0.9901	0.0292	0.9868	0.0631

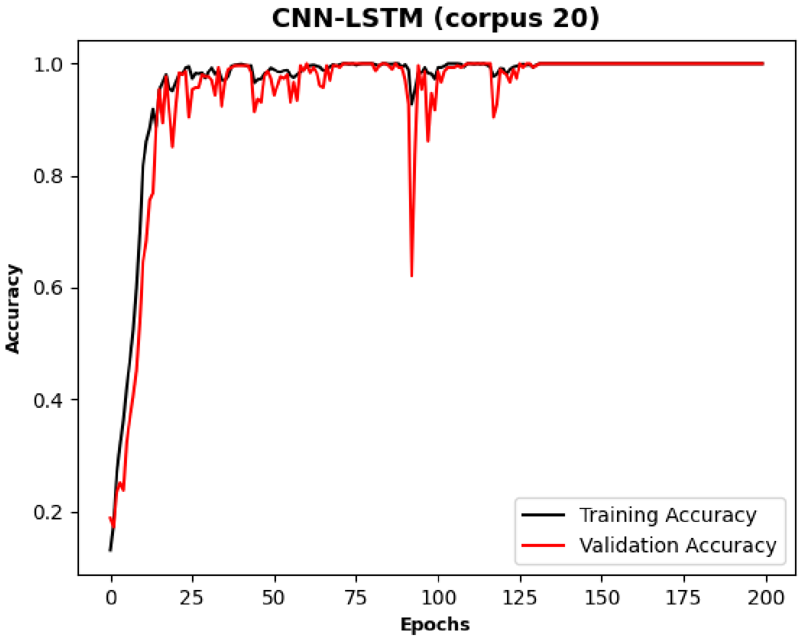


Figure 7-5 CNN-LSTM (corpus 20) training & validation accuracy through epochs

Table 6 CNN-LSTM hybrid with DTW (corpus 20) model progression for the first 60 epochs

CNN LSTM Hybrid with DTW – 20 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.2294	2.3891	0.0891	3.6053
2	0.4356	1.6404	0.2343	2.5832
3	0.5404	1.3523	0.3696	2.3280
5	0.7442	0.7723	0.5380	1.7241
8	0.8465	0.4507	0.6172	1.3059
10	0.8927	0.3140	0.5215	2.4957
20	0.0814	0.9711	0.9637	0.0900
30	0.9455	0.1534	0.9670	0.0780
40	0.9901	0.0306	0.9868	0.0404
50	0.9389	0.2451	0.7393	1.1305
60	0.9959	0.0169	1.0000	0.0043

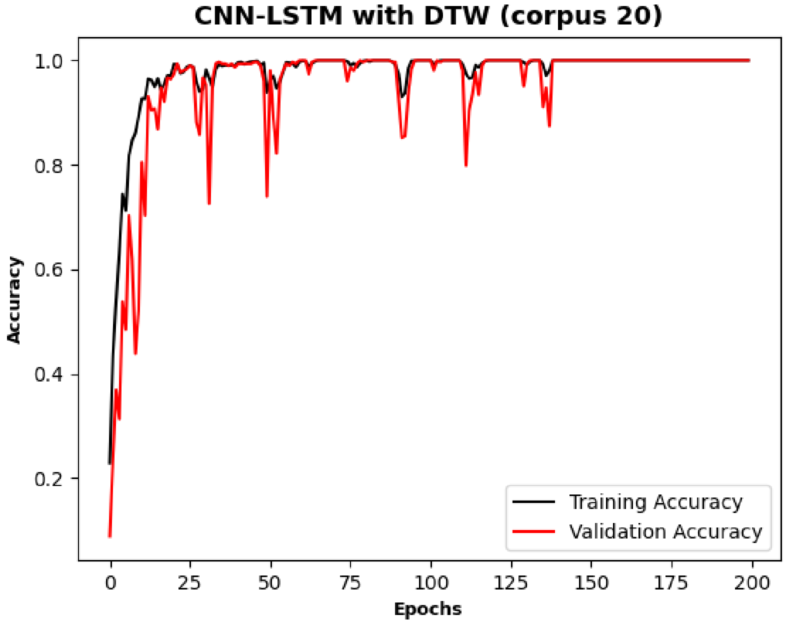


Figure 7-6 CNN-LSTM with DTW (corpus 20) training & validation accuracy through epochs

7.2. Corpus 100

This corpus consists of the first 100 glosses of the WLASL dataset. On average every gloss has 13 video instances assigned to it. The following tables present the progression of each model during epochs.

The results follow the same patter with the previous corpus. Specifically, the CNN models preformed outstandingly, as they reached 97% accuracy 3 epochs faster and with very few drops (Figures 7-7 & 7-8), while the hybrid models had smoother progressions in their training (Figures 7-11 & 7-12).

The LSTM model was affected most by the increased size of glosses. Despite, the low accuracy of 60% of the LSTM model without DTW, the LSTM on data with DTW reached 97% accuracy (Figures 7-9 & 7-10). Its learning curve was steeper, but after the 110th epoch the accuracy increased dramatically. The CNN ones are still performing better, but at least it is an acceptable high performing model.

From the above, it is obvious that the increased number of available sequences helped the models to better recognize the differences between gestures, and become more robust.

Table 7 CNN (corpus 100) model progression for the first 80 epochs

CNN – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0362	4.4799	0.0398	4.2844
2	0.1210	3.6951	0.0934	3.8359
3	0.3549	2.5370	0.4342	2.2566
5	0.8871	0.4292	0.9051	0.3667
8	0.9778	0.0829	0.9701	0.1159
10	0.9770	0.0856	0.9655	0.1396
20	0.9812	0.0611	0.9793	0.0701
30	0.9759	0.0907	0.9655	0.1514

40	0.9855	0.0606	0.9648	0.1699
50	0.9900	0.0236	0.9870	0.0633
60	0.9872	0.0320	0.9870	0.0465
70	0.9858	0.0408	0.9617	0.1599
80	0.9879	0.0394	0.9908	0.0427

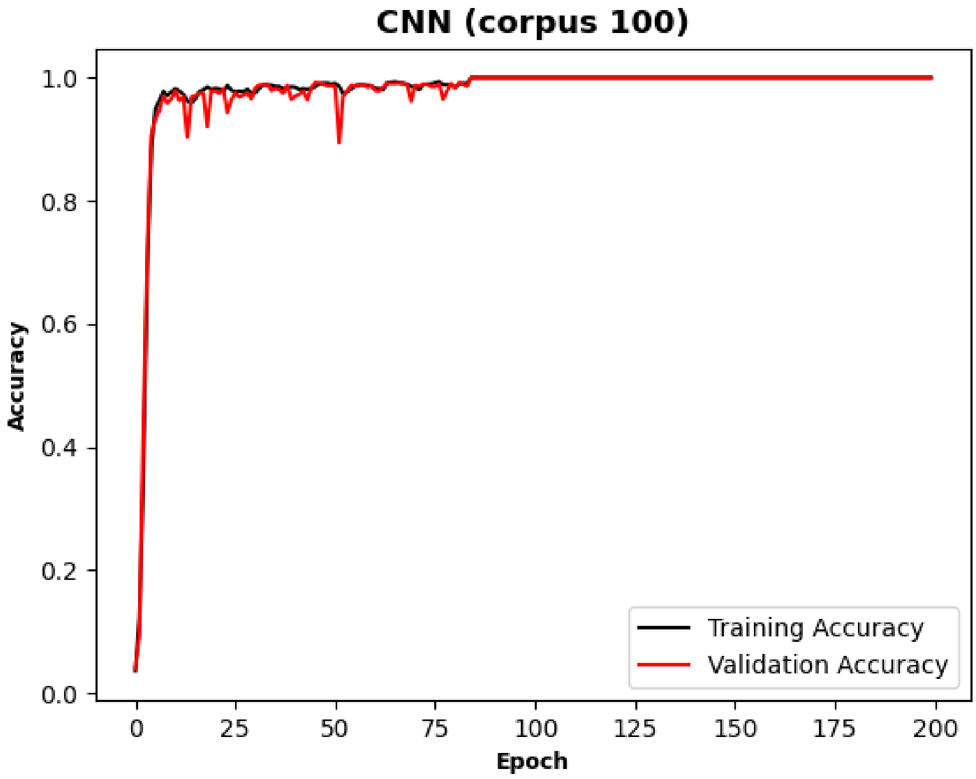


Figure 7-7 CNN (corpus 100) training & validation accuracy through epochs

Table 8 CNN with DTW (corpus 100) model progression for the first 40 epochs

CNN with DTW – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.2132	3.4716	0.0881	4.2459
2	0.6558	1.2564	0.3484	2.4963
3	0.8947	0.3708	0.7496	0.8102
5	0.9753	0.0991	0.9832	0.0826
8	0.9908	0.0375	0.9877	0.0744
10	0.9902	0.0425	0.9832	0.0894
20	0.9883	0.0446	0.9732	0.0828
30	0.9952	0.0274	0.9900	0.0193
40	0.9939	0.0285	0.9778	0.1113

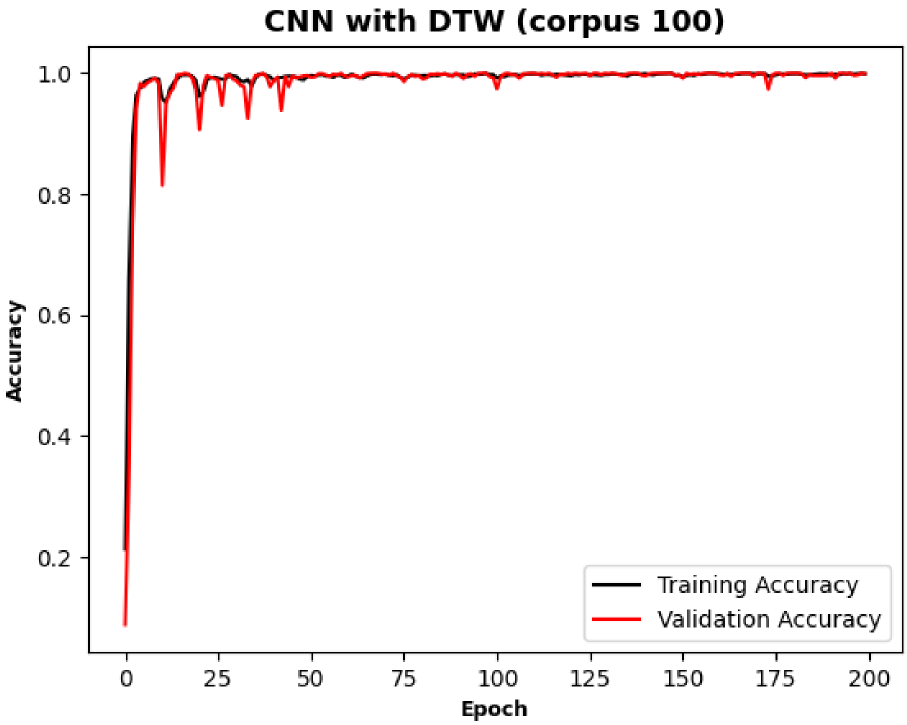


Figure 7-8 CNN with DTW (corpus 100) training & validation accuracy through epochs

Table 9 LSTM (corpus 100) model progression for the first 200 epochs

LSTM – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0115	4.6029	0.0207	4.5898
10	0.0165	4.5618	0.0214	4.5495
20	0.0188	4.5713	0.0245	4.5702
30	0.0193	4.5682	0.0253	4.5695
40	0.0190	4.5719	0.0260	4.5705
50	0.0205	4.5208	0.0299	4.5095
60	0.0310	4.4033	0.0337	4.4030
70	0.0394	4.1466	0.0314	4.1003
80	0.0725	3.7432	0.0574	3.7668
90	0.1083	3.5351	0.0735	3.6223
100	0.1575	3.2926	0.1279	3.4206
110	0.2127	2.9832	0.1769	3.1373
120	0.2693	2.7142	0.2297	2.9218
130	0.3063	2.5454	0.2274	2.7904
140	0.3524	2.3103	0.3101	2.5152
150	0.3991	2.1346	0.3446	2.3725
160	0.4453	1.9503	0.3859	2.1674
170	0.5036	1.7233	0.4495	1.9785
180	0.4966	1.7061	0.4250	2.0050
190	0.5549	1.4765	0.4296	2.1668
200	0.5570	1.4833	0.5138	1.7210

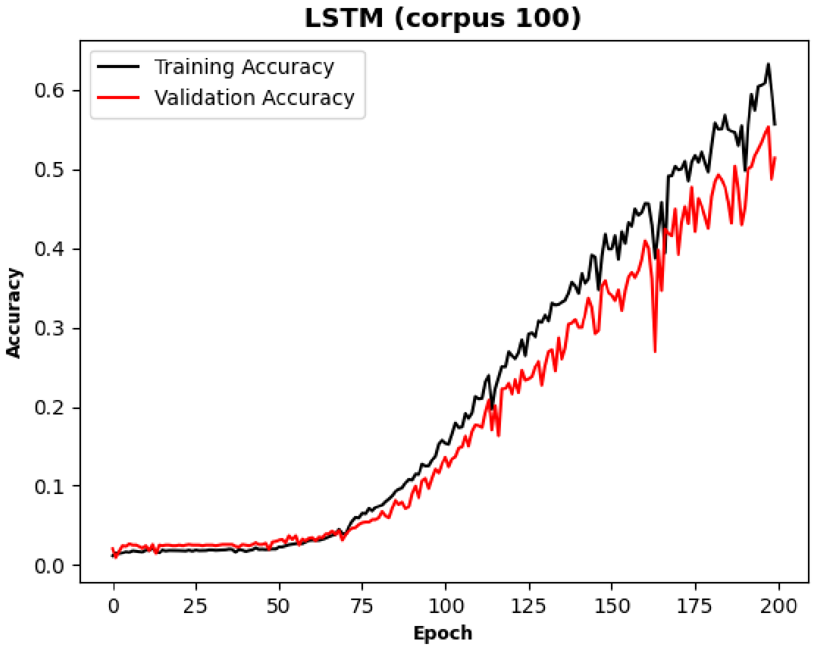


Figure 7-9 LSTM (corpus 100) training & validation accuracy through epochs

Table 10 LSTM with DTW (corpus 100) model progression for the first 200 epochs

LSTM with DTW – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0126	4.6095	0.0130	4.6011
10	0.0741	3.8015	0.0819	3.7586
20	0.1453	3.4181	0.1577	3.4072
30	0.2085	2.9780	0.2228	3.0019
40	0.2996	2.2986	0.3055	2.3851
50	0.4391	1.7638	0.4089	1.8624
60	0.5572	1.3379	0.4740	1.7107
70	0.7081	0.8792	0.6302	1.0868
80	0.7841	0.6709	0.7511	0.7458
90	0.8645	0.4277	0.8162	0.5694
100	0.8834	0.3524	0.8645	0.3934

110	0.8756	0.3713	0.8806	0.3321
120	0.9213	0.2550	0.9227	0.2263
130	0.9378	0.1999	0.9311	0.1963
140	0.9755	0.0793	0.9640	0.1273
150	0.9537	0.1519	0.9502	0.1627
160	0.9889	0.0449	0.9832	0.0610
170	0.9839	0.0588	0.9479	0.1574
180	0.9598	0.1281	0.9502	0.1565
190	0.9366	0.1887	0.9510	0.1399
200	0.9617	0.1351	0.9525	0.1644

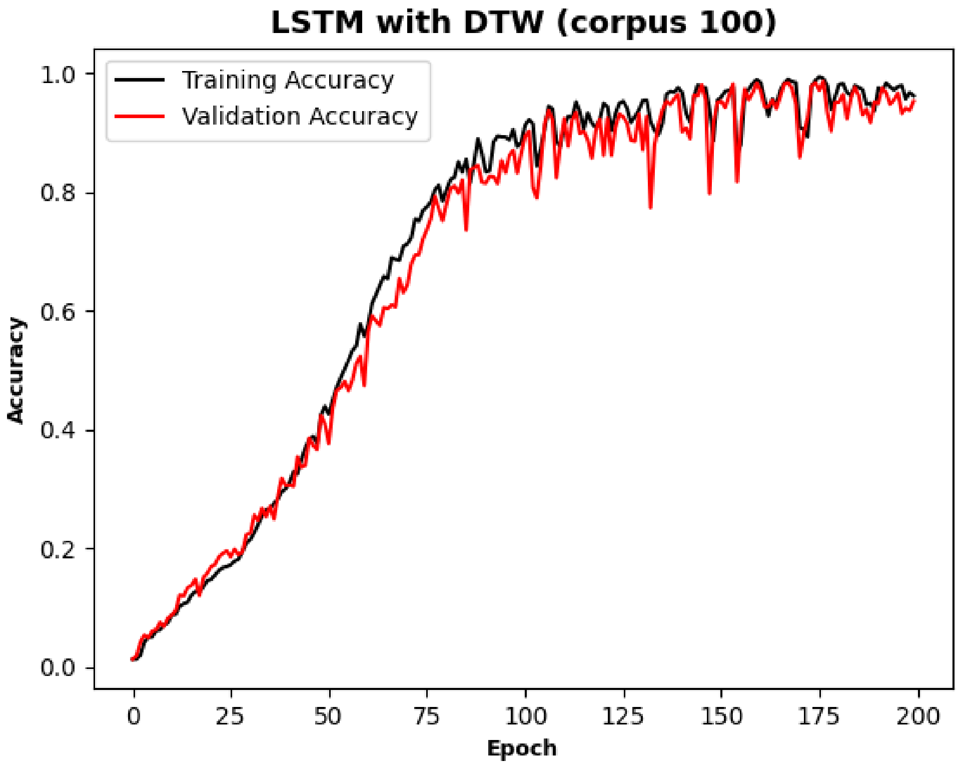


Figure 7-10 LSTM with DTW (corpus 100) training & validation accuracy through epochs

Table 11 CNN LSTM Hybrid (corpus 100) model progression for the first 80 epochs

CNN LSTM Hybrid – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0366	4.1836	0.0268	4.5083
2	0.0963	3.6325	0.1179	3.5407
3	0.1771	3.1573	0.1662	3.1150
5	0.4772	1.8091	0.4548	1.8315
8	0.8520	0.4589	0.8499	0.4832
10	0.9229	0.2566	0.8744	0.3858
15	0.9611	0.1346	0.9250	0.2359
20	0.9357	0.2166	0.9579	0.1108
30	0.9726	0.0862	0.9518	0.1629
40	0.9786	0.0661	0.9900	0.0619
50	0.9941	0.0142	0.9900	0.0432
60	0.9922	0.0165	0.9900	0.0436
70	0.9950	0.0102	0.9923	0.0448
80	0.9966	0.0077	0.9916	0.0360

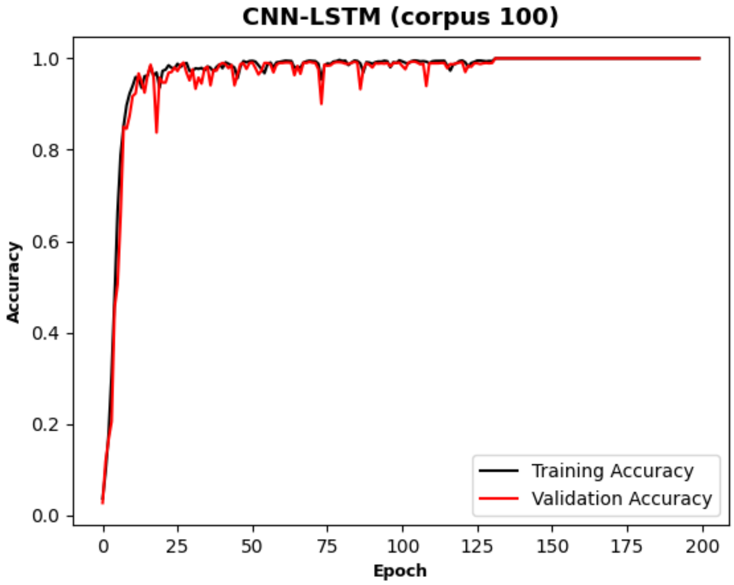


Figure 7-11 CNN-LSTM (corpus 100) training & validation accuracy through epochs

Table 12 CNN LSTM Hybrid (corpus 100) model progression for the first 40 epochs

CNN LSTM Hybrid with DTW – 100 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.1501	3.5693	0.1194	4.1306
2	0.3742	2.2424	0.3277	2.2880
3	0.5630	1.4627	0.4931	1.6392
5	0.8248	0.5393	0.7680	0.6915
8	0.9521	0.1581	0.9372	0.2435
10	0.9468	0.1537	0.9686	0.1097
15	0.9590	0.1398	0.9686	0.1315
20	0.9941	0.0245	0.9832	0.0805
30	0.9914	0.0297	0.9709	0.1062
40	0.9920	0.0264	0.9954	0.0105

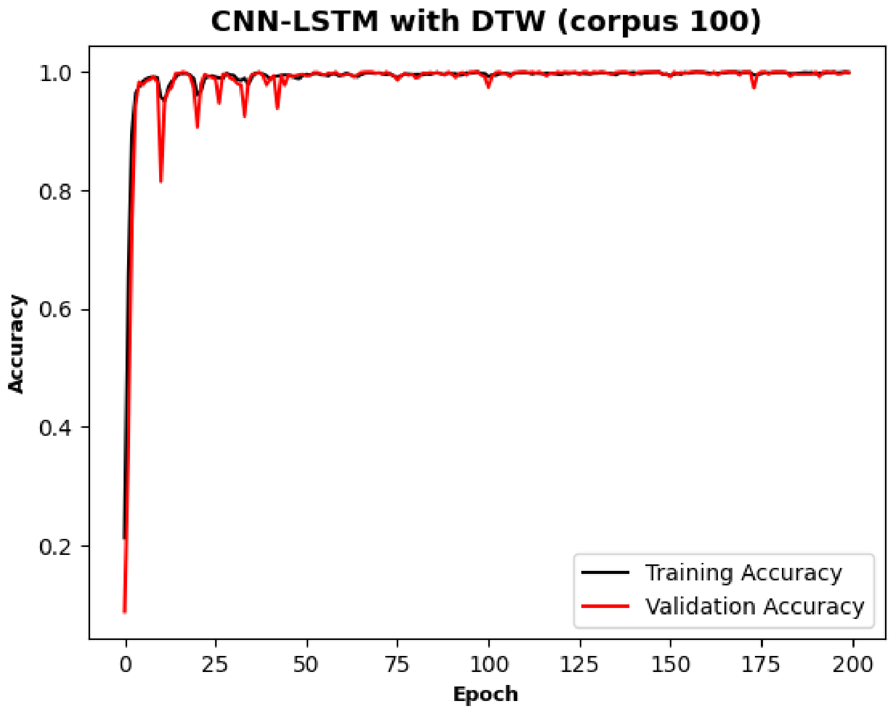


Figure 7-12 CNN-LSTM with DTW (corpus 100) training & validation accuracy through epochs

7.3. Corpus 300

This corpus consists of the first 300 glosses of the WLASL dataset. On average every gloss has 9 video instances assigned to it. The following tables present the progression of each model during epochs.

In the largest corpus, the effect of DTW diminishes even more for the CNN models. In particular, DTW helps the validation accuracy, and makes the learning curve smoother; it does not actually affects the accuracy or how fast the CNN models learn.

The only exception is the LSTM model. For 300 different words, the LSTM model never actually converged. The best accuracy it achieved was 25%. However, DTW tripled the LSTM accuracy as shown in Figure 7-16. The LSTM with DTW had 70% accuracy with an almost linear leaning progression.

Table 13 CNN (corpus 300) model progression for the first 80 epochs

CNN – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0096	5.552	0.0148	5.2515
2	0.0296	4.953	0.0464	4.8157
3	0.0752	4.4433	0.1088	4.2116
5	0.3211	2.783	0.3523	2.6685
8	0.8302	0.5699	0.7832	0.7752
10	0.9093	0.2958	0.935	0.197
15	0.9551	0.1368	0.9636	0.1173
20	0.9552	0.139	0.9651	0.0969
30	0.9698	0.0954	0.9729	0.0691
40	0.9734	0.0905	0.9592	0.1523
50	0.9787	0.0615	0.9762	0.0905
60	0.9813	0.0499	0.984	0.0418
70	0.9817	0.0526	0.9837	0.0605
80	0.9866	0.0318	0.9833	0.0329

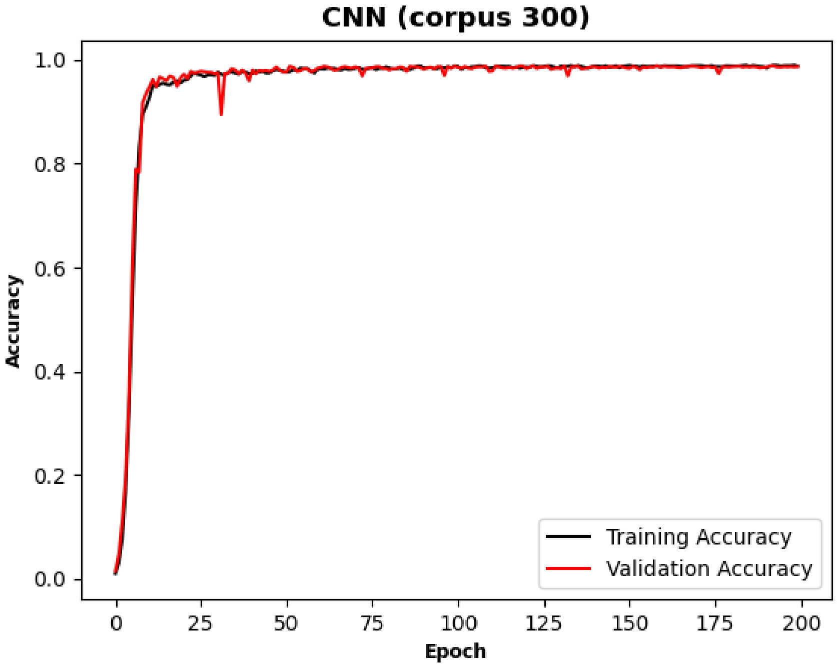


Figure 7-13 CNN (corpus 300) training & validation accuracy through epochs

Table 14 CNN with DTW (corpus 300) model progression for the first 40 epochs

CNN with DTW – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0512	5.0026	0.1125	4.172
2	0.294	3.0437	0.3255	2.8904
3	0.6085	1.5075	0.703	1.0886
5	0.9275	0.2597	0.9532	0.1649
8	0.9692	0.111	0.9662	0.1066
10	0.9578	0.1492	0.9666	0.1112
20	0.9752	0.0845	0.9755	0.0627
30	0.9847	0.0547	0.9829	0.0706
40	0.9878	0.0399	0.9777	0.0618

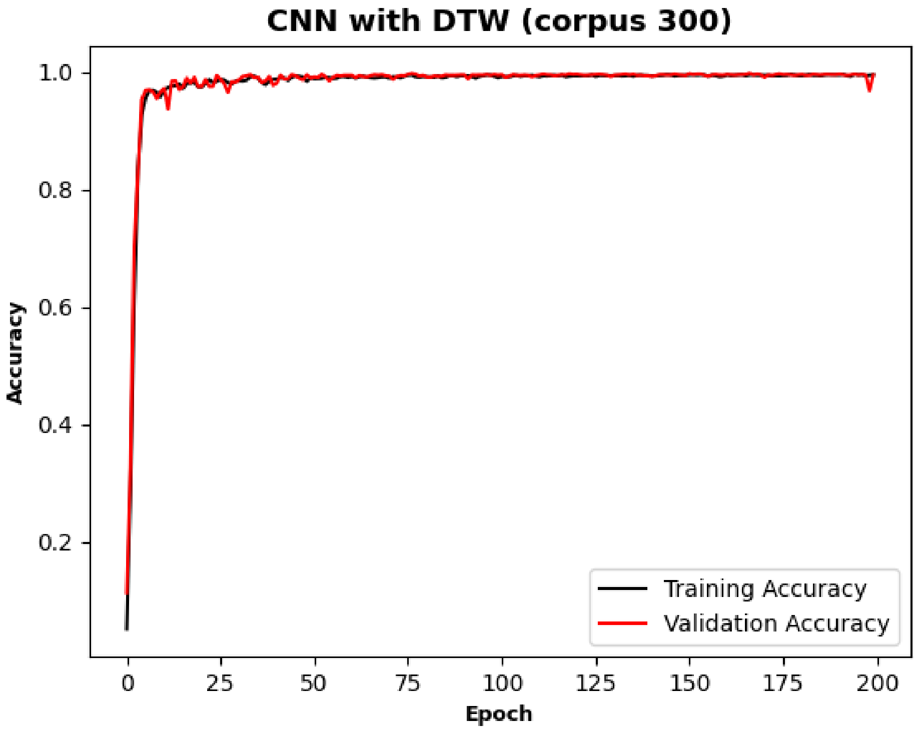


Figure 7-14 CNN with DTW (corpus 300) training & validation accuracy through epochs

Table 15 LSTM (corpus 300) model progression for the first 200 epochs

LSTM – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0031	5.7042	0.0030	5.7035
10	0.0114	5.4124	0.0122	5.4596
20	0.0289	4.7915	0.0234	4.8842
30	0.0361	4.4683	0.0256	4.5956
40	0.0417	4.3417	0.0312	4.4222
50	0.0542	4.1069	0.0397	4.2604
60	0.0738	3.8977	0.0538	4.0627
70	0.0924	3.709	0.0824	3.8237
80	0.109	3.5007	0.0883	3.7162

90	0.1242	3.3671	0.0820	3.6693
100	0.1332	3.2355	0.1229	3.3472
110	0.1334	3.1953	0.1225	3.2488
120	0.1594	2.9878	0.1173	3.2438
130	0.1637	2.9535	0.1180	3.2894
140	0.1747	2.8658	0.1600	2.9622
150	0.1779	2.8504	0.1299	3.0088
160	0.191	2.7154	0.1615	2.8746
170	0.2059	2.6227	0.1704	2.7938
180	0.2167	2.5551	0.1578	2.8011
190	0.2106	2.5854	0.1711	2.7232
200	0.2447	2.3978	0.1960	2.6141

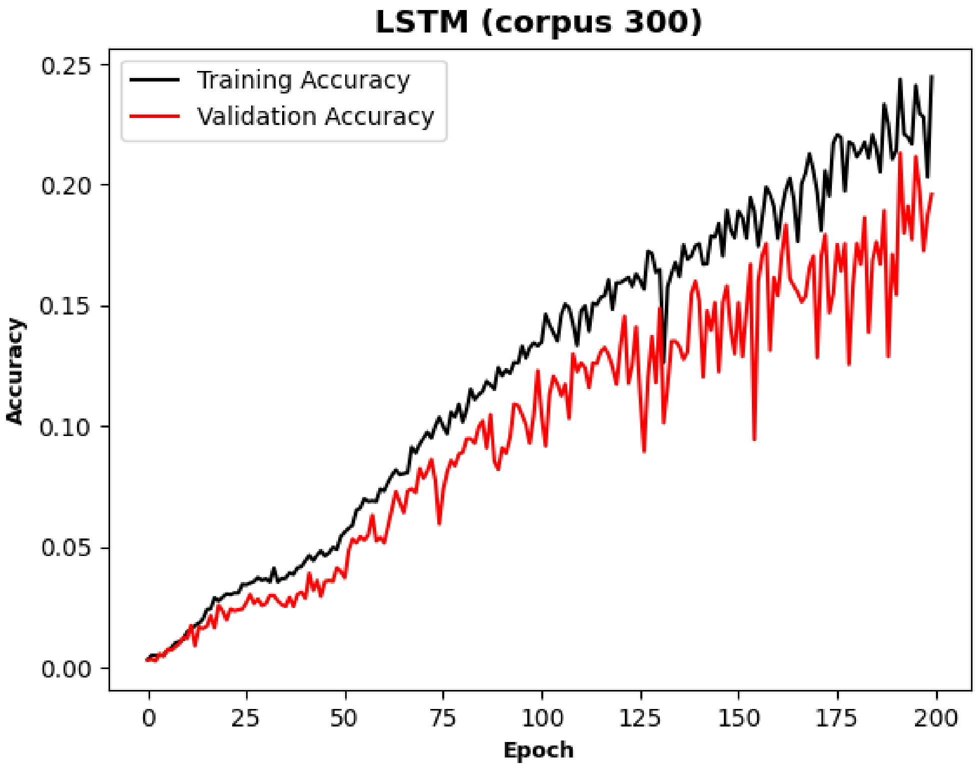


Figure 7-15 LSTM (corpus 300) training & validation accuracy through epochs

Table 16 LSTM with DTW (corpus 300) model progression for the first 300 epochs

LSTM with DTW – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0042	5.7054	0.0071	5.7036
10	0.0491	4.5179	0.0627	4.5584
20	0.1358	3.3169	0.1177	3.4997
30	0.192	2.801	0.1663	3.0356
40	0.2296	2.5199	0.1975	2.7066
50	0.3038	2.1407	0.2398	2.4226
60	0.337	1.9513	0.2635	2.2273
70	0.3765	1.8151	0.333	1.9893
80	0.4303	1.6076	0.376	1.771
90	0.4552	1.5202	0.4176	1.8267
100	0.4698	1.4744	0.4621	1.4912
110	0.5445	1.248	0.4402	1.5521
120	0.5586	1.2004	0.5434	1.2858
130	0.5793	1.1439	0.5568	1.2143
140	0.5326	1.3556	0.5431	1.2827
150	0.5961	1.1325	0.5798	1.1615
160	0.614	1.0638	0.5805	1.1494
170	0.6346	0.9918	0.6121	1.0539
180	0.6687	0.9065	0.585	1.1499
190	0.6851	0.8656	0.6414	0.9961
200	0.7276	0.7268	0.6247	1.0445

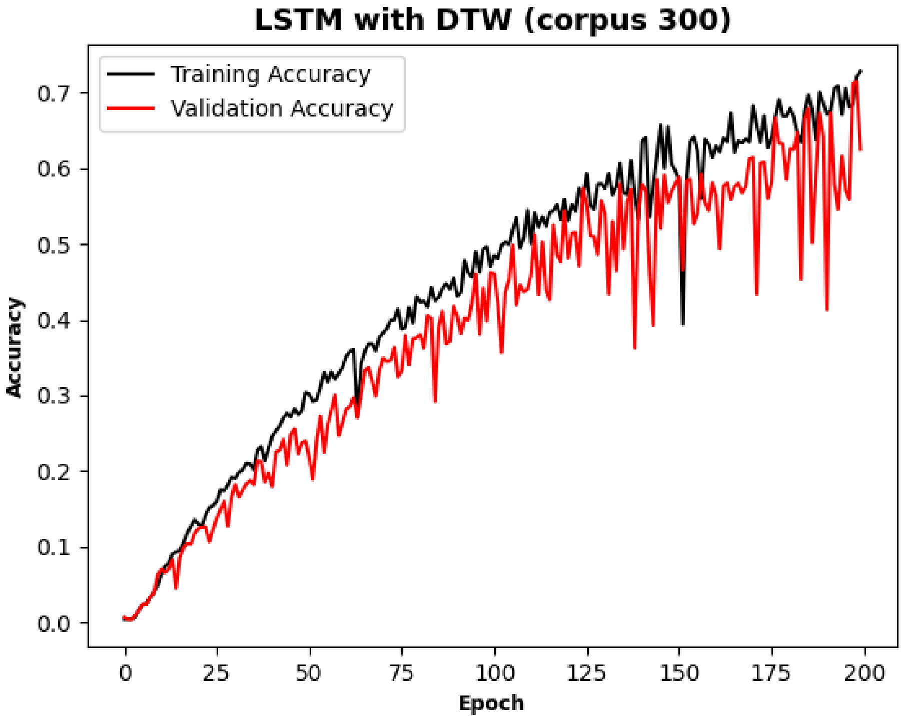


Figure 7-16 LSTM with DTW (corpus 300) training & validation accuracy through epochs

Table 17 CNN LSTM Hybrid (corpus 300) model progression for the first 40 epochs

CNN LSTM Hybrid – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0097	5.2716	0.0119	5.1137
2	0.0454	4.6703	0.0646	4.3555
3	0.1463	3.7019	0.1381	3.6195
5	0.5098	1.7577	0.5427	1.636
8	0.786	0.688	0.6626	1.1552
10	0.848	0.4716	0.8422	0.5226
15	0.9166	0.2469	0.9143	0.2483
20	0.9395	0.1755	0.9529	0.1494
30	0.9669	0.0988	0.9681	0.091
40	0.9754	0.0704	0.977	0.0668

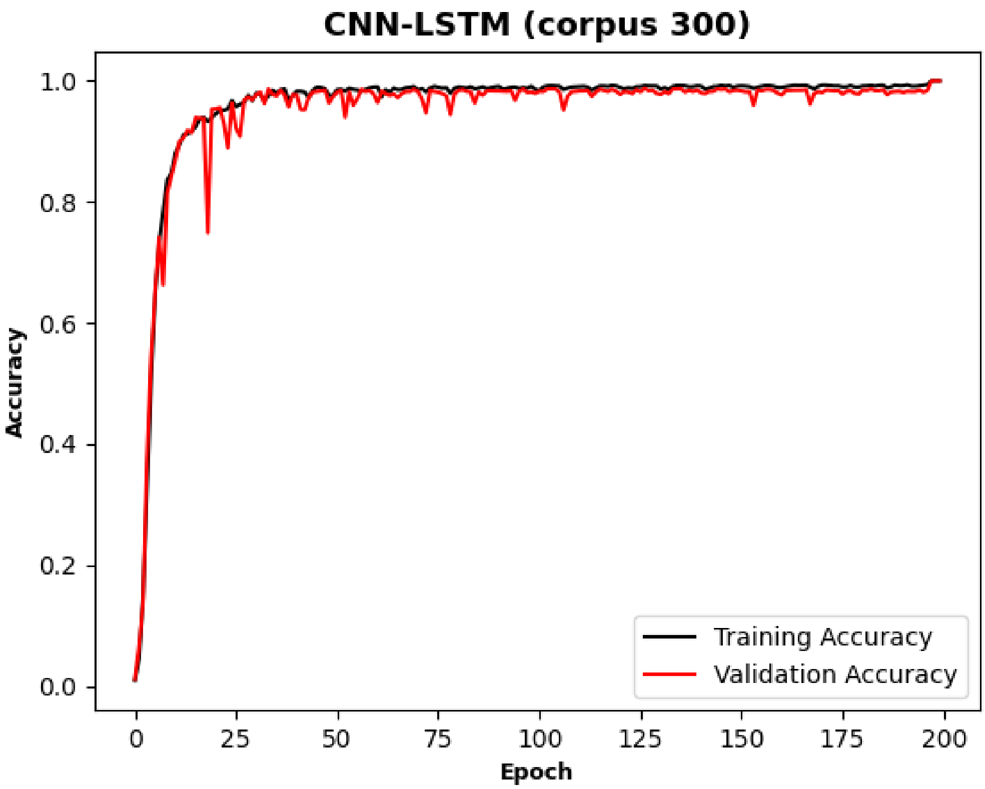


Figure 7-17 CNN-LSTM (corpus 300) training & validation accuracy through epochs

Table 18 CNN LSTM Hybrid (corpus 300) model progression for the first 40 epochs

CNN LSTM Hybrid with DTW – 300 glosses				
Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1	0.0428	4.9058	0.0857	4.2725
2	0.2227	3.3235	0.3129	2.7486
3	0.4552	2.0214	0.5019	1.7927
5	0.8118	0.5962	0.8523	0.4612
8	0.9442	0.1848	0.9495	0.1581
10	0.9545	0.1451	0.9699	0.1033

15	0.9759	0.0734	0.9837	0.0503
20	0.9773	0.0679	0.9774	0.0942
30	0.9907	0.0303	0.9918	0.021
40	0.9846	0.0493	0.9807	0.0764

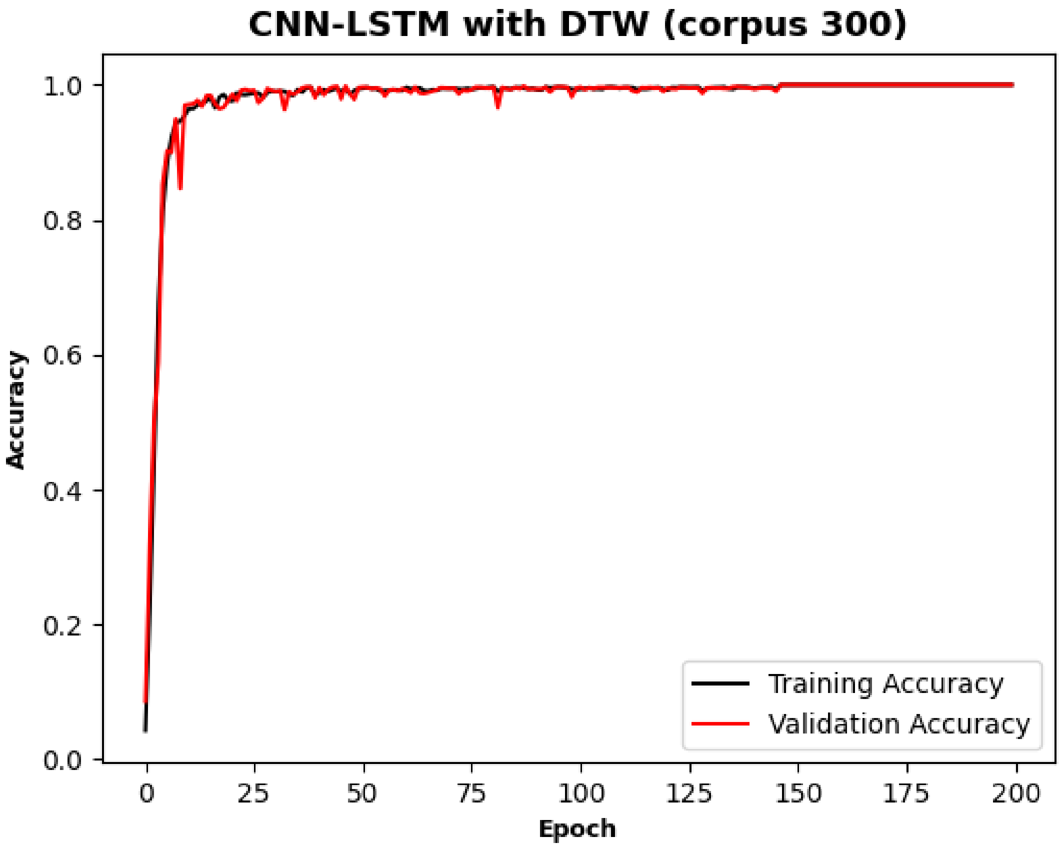


Figure 7-18 CNN-LSTM with DTW (corpus 300) training & validation accuracy through epochs

7.4. Comparative Results

In this chapter we compare the accuracy of the proposed models for every corpus to better visualize the performance of the different architectures. The following diagrams present the accuracy of the model across 200 epochs.

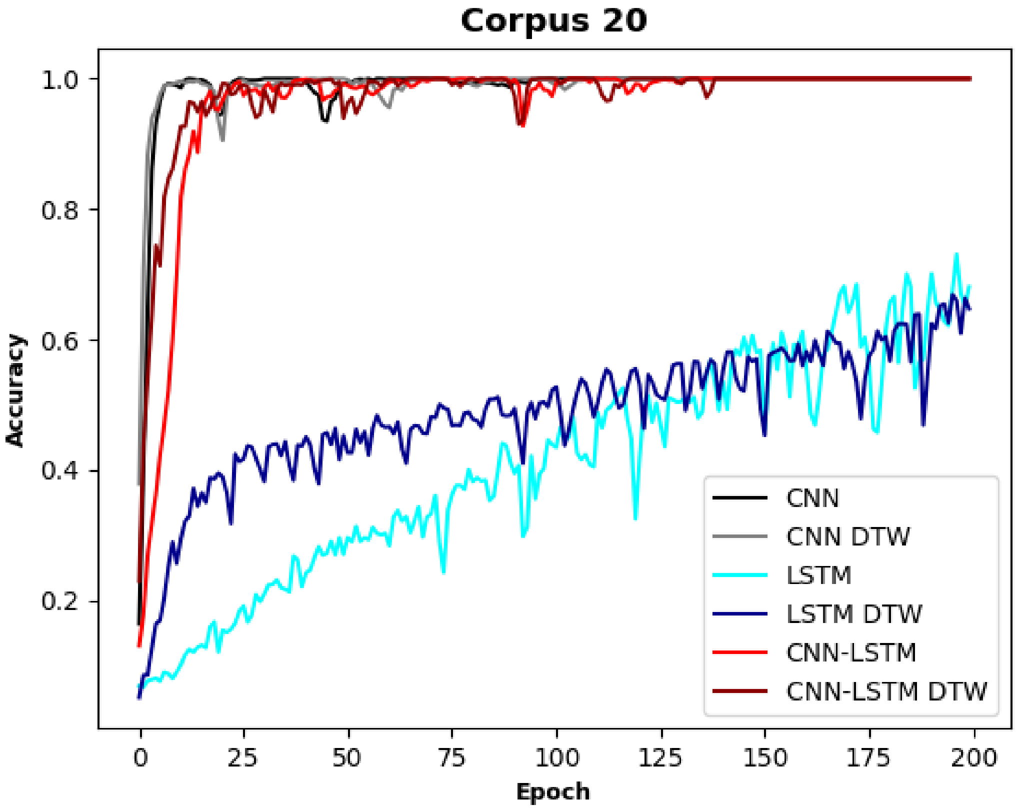


Figure 7-19 Corpus 20 all models validation accuracy through epochs

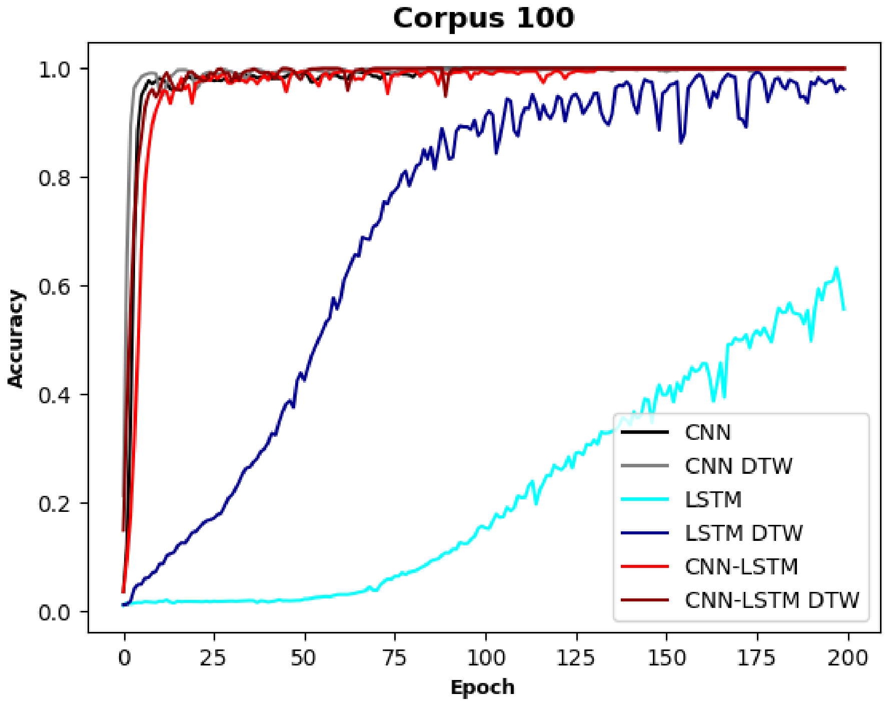


Figure 7-20 Corpus 100 all models validation accuracy through epochs

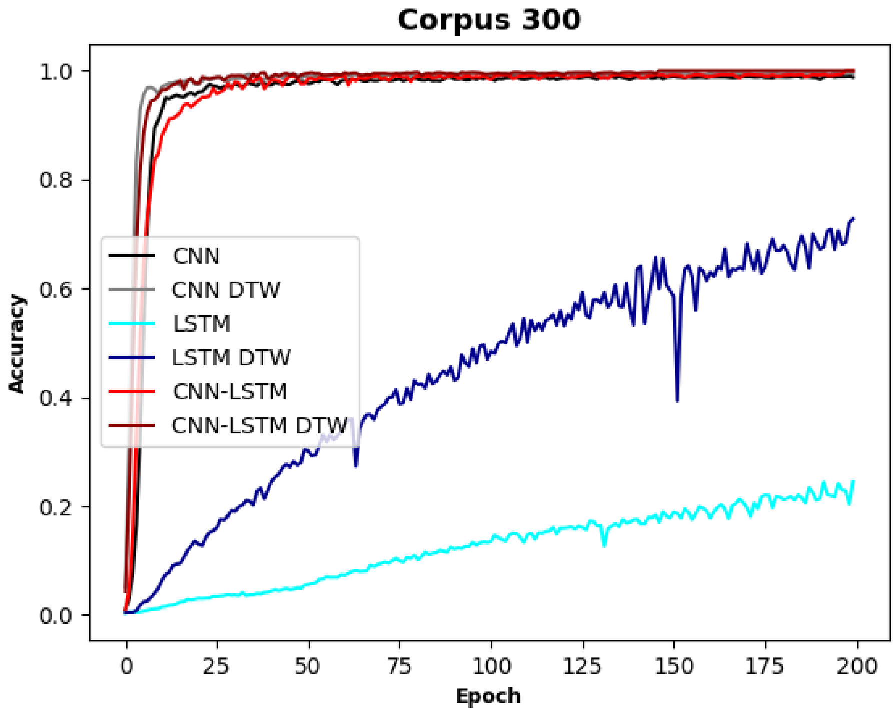


Figure 7-21 Corpus 300 all models validation accuracy through epochs

Finally, we selected the best performing models across all corpuses based on the best achieved accuracy and the minimum number of epochs required to hit 90% accuracy. The following figure shows the progression of accuracy for the first 50 epochs.

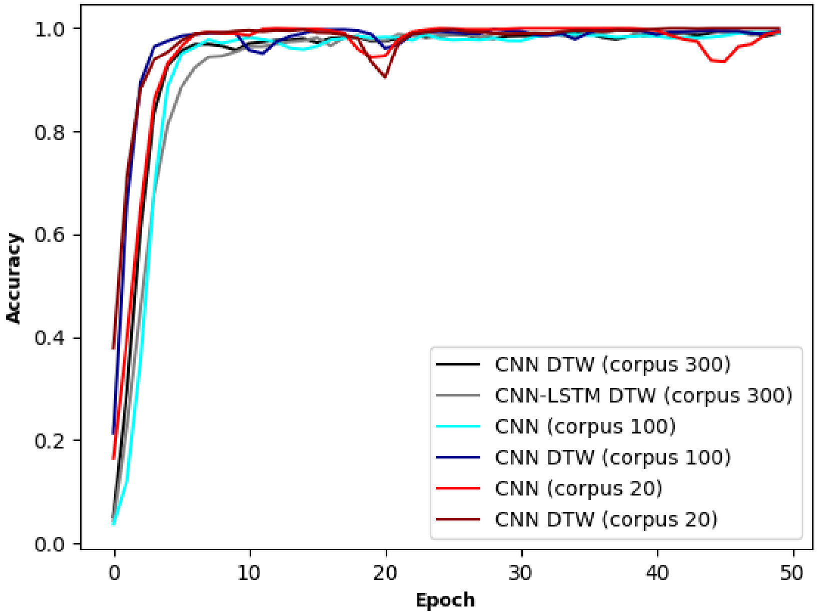


Figure 7-22 Best performing models by validation accuracy the first 50 epochs

8. Conclusion and Future Scope

Sign Language Recognition is a research topic with both social and academic interest. Every region has its own Sign Language, with different gestures and meanings. Machine Learning can help decode these sings and sequences of signs with a common methodology.

In this project we proposed a robust SLR model that can accurately predict sign language words without the need for specific equipment. For that purpose, we processed the video sequences by applying various data augmentation techniques, and then extracted hand and pose landmarks. The landmarks were extracted from a corpus of 20, 100 and 300 glosses. Each of these sets of landmarks were used as input into three proposed architectures; a Convolutional Neural Network, a Long Short-Term Memory Recurrent Neural Network, and a hybrid of those two. Additionally, a Dynamic Time Wrapping technique was used to attempt synchronization between the gestures of the same gloss in an effort to help our models perform better.

The CNN and CNN-LSTM hybrid architectures performed exceptionally reaching 98% accuracy across all corpuses. DTW assisted the aforementioned models to reduce the training time. On the other hand, the proposed LSTM architecture did not perform well, except for the LSTM model with DTW for the corpus of size 100. Nonetheless, we achieved our goal of performing time series analysis and producing an accurate SLR model which only requires a camera and some computational power.

In the future, continuous sign language recognition will be considered. The challenge there is threefold. Firstly, a large enough dataset must be created, where the same words are used in different sentences, and the number of instances of each sequence are evenly distributed. Secondly, the model should be able to recognize accurately these sentences. Thirdly, the model should be able to recognize the individual components-words. This can be achieved by splitting the video instances per word, or ideally create an architecture where that is not needed.

References

[1] I. Achmed, "*Upper body pose recognition and estimation towards the translation of South African Sign Language*," Master's thesis, University of the Western Cape, Computer Science, 2010.

[2] D. Brown, "Faster Upper Body Pose Estimation and Recognition using CUDA," Master's thesis, University of the Western Cape, Computer Science, 2012.

[3] A. Maruch, "Talking with the hearing-impaired," January 2010, [Online] Available at http://www.deafsa.co.za/index-2.html.

[4] S. Prillwitz: HamNoSys. Version 2.0; Hamburg Notation System for Sign Language. An Introduction Guide. In Signum Verlag, 1989.

[5] S. Liddell: Holds and Positions: Comparing Two Models of Segmentation in ASL. In Phonetics and Phonology: Current Issues in ASL Phonology, pp. 189–211. Academic Press Inc., 1993.

[6] Y. Cui, D. Swets, J. Weng: Learning-Based Hand Sign Recognition Using SHOSLIF-M. In In Proceeding of Int. Workshop on Automatic Face and Gesture Recognition, pp. 201–206, Zurich, 1995.

[7] M.Z. F. Quek: Inductive Learning in Hand Pose Recognition. In Proc. of Second IEEE Int. Conf. on Automatic Face and Gesture Recognition, Washington, DC, USA, 1996. IEEE.

[8] J. Triesch, C. von der Malsburg: A System for Person-Independent Hand Posture Recognition against Complex Backgrounds. IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 23, No. 12, pp. 1449–1453, Dec. 2001.

[9] T. Deselaers, D. Keysers, H. Ney: Improving a Discriminative Approach to Object Recognition using Image Patches. In DAGM 2005, Pattern Recognition, 26th DAGM Symposium, Lecture Notes in Computer Science, pp. 326–333, Vienna, Austria, Aug. 2005.

[10] H. Birk, T. Moeslund, C. Madsen: Real-Time Recognition of Hand Alphabet Gestures Using Principal Component Analysis. In Proc. of the 10th Scandinavian Conference on Image Analysis, Laeenranta, Finland, June 1997.

[11] S. Mehdi, Y. Khan: Sign Language Recognition Using Sensor Gloves. In Proc. of the 9th International Conference on Neural Information Processing, Vol. 5, pp. 2204–2206, Singapore, 2002.

[12] K. Abe, H. Saito, S. Ozawa: Virtual 3D Interface System via Hand Motion Recognition From Two Cameras. IEEE Trans. Systems, Man, and Cybernetics, Vol. 32, No. 4, pp. 536–540, July 2002.

[13] S. Malassiottis, N. Aifanti, M. Strintzis: A Gesture Recognition System Using 3D Data. In Proc. IEEE 1st International Symposium on 3D Data Processing, Visualization, and Transmission, Vol. 1, pp. 190–193, Padova, Italy, June 2002.

[14] J. Hernandez-Rebollar, R. Lindeman, N. Kyriakopoulos: A Multi-Class Pattern Recognition System for Practical Finger Spelling Translation. In Proc. Of the 4th IEEE International Conference on Multimodal Interfaces, pp. 185–190, Pittsburgh, PA, Oct. 2002.

[15] S. Akyol, U. Canzler, K. Bengler, W. Hahn: Gesture Control for Use in Automobiles. In Proc. IAPR Workshop Machine Vision Applications, pp. 349–352, Tokyo, Japan, Nov. 2000.

[16] A.Pelkmann. Entwicklung eines Klassifikators zur videobasierten Erkennung von Gesten. Studienarbeit, Diploma thesis, RWTH Aachen University, Aachen, Germany, feb 1999.

[17] P. Dreuw. Appearance-Based Gesture Recognition. Diploma thesis, Lehrstuhl f¨ur Informatik VI, RWTH Aachen University, Aachen, Jan. 2005.

[18] P. Dreuw, D. Keysers, T. Deselaers, H. Ney: Gesture Recognition Using Image Comparison Methods. In GW 2005, 6th Int. Workshop on Gesture in Human-Computer Interaction and Simulation, Vannes, France, May 2005.

[19] P. Dreuw, T. Deselaers, D. Keysers, H. Ney: Modeling Image Variability in Appearance-Based Gesture Recognition. In ECCV 2006 3rd Workshop on Statistical Methods in Multi-Image and Video Processing, pp. 7–18, Graz, Austria, May 2006.

[20] M.W. Kadous: Machine Recognition of Auslan Signs using Powergloves: Toward Largelexicon Recognition of Sign Language. In Proc. Workshop Integration Gesture Language Speech, pp. 165–174, Newark, Delaware, Oct. 1996.

[21] H. Matsuo, S. Igi, S. Lu, Y. Nagashima, Y. Takata, T. Teshima: The Recognition Algorithm with Noncantact for Japanese Sign Language Using Morphological Analysis. In Proc. Int. Gesture Workshop, pp. 273–284, Bielefeld, Germany, Sept. 1997.

[22] G. Rigoll, A. Kosmala: New Improved Feature ExtractionMethods for Real-Time High Performance Image Sequence Recognition. In IEEE Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 2901–2904, Munich, 1997.

[23] S. Eickeler, A. Kosmala, G. Rigoll: Hidden Markov Model Based Continuous Online Gesture Recognition. In Int. Conference on Pattern Recognition (ICPR), pp. 1206–1208, Brisbane, 1998.

[24] R.H. Liang, M. Ouhyoung: A Real-time Continuous Gesture Recognition System for Sign Language. In Proc. 3rd IEEE Int. Conf. on Automatic Face and Gesture Recognition, pp. 558–565, Nara, Japan, April 1998. IEEE.

[25] C. Vogler, D. Metaxas: Adapting Hidden Markov Models for ASL Recognition by Using Three dimentional Computer Vision Methods. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 156–161, Orlando, FL, 1997.

[26] C. Vogler, D. Metaxas: Parallel Hidden Markov Models for American Sign Language Recognition. In Proceedings of the International Conference on Computer Vision, Kerkyra, Greece, sep 1999.

[27] T. Starner, J. Weaver, A. Pentland: Real-time American Sign Language Recognition Using Desk and Wearable Computer Based Video. Transaction of Pattern Analysis and Machine Intelligence, Vol. 20(2), pp. 1371–1375, 1998.

[28] B. Bauer, H. Hienz: Relevant Features for Video-based Continuous Sign Language Recognition. In Proceedings of the 4th International Conference Automatic Face and Gesture Recognition, pp. 440–445, Grenoble, France, march 2000.

[29] B. Bauer, H. Hienz, K. Kraiss: Video-Based Continuous Sign Language Recognition Using StatisticalMethods. In Proceedings of the International Conference on Pattern Recognition, pp. 463–466, Barcelona, Spain, 2000.

[30] R. Bowden, D. Windridge, T. Kabir, A. Zisserman, M. Bardy: A Linguaistic Feature Vector for the Visual Interpretation of Sign Language. In Proceedings of ECCV 2004, the 8th European Conference on Computer Vision, Vol. 1, pp. 391–401, Prague, Czech Republic, 2004.

[31] G. Fang, W. Gao, D. Zhao: Large Vocabulary Sign Language Recognition Base on Fuzzy Decision Trees. IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 34(3), pp. 305–314, 2004.

[32] U.M. Erdem, S. Sclaroff: Automatic Detection of Relevant Head Gestures in American Sign Language Communication. In Proceedings of the International Conference on Pattern Recognition, ICPR, Qubec, Canada, Aug. 2002.

[33] H. Kashima, H. Hongo, K. Kato, K. Yamamoto: A Robust Iris Detection Method of Facial and Eye Movement. In VI2001, Vision Interface Annual Conference, Ottawa, Canada, June 2001.

[34] U. Canzler, T. Dziurzyk: Extraction of Non Manual Features for Videobased Sign Language Recognition. In Proc. IAPR Workshop Machine Vision Applications, pp. 318–321, 2002.

[35] C. Vogler, S. Goldenstein: Analysis of Facial Expressions in American Sign Language. In Proceedings of the 3rd Intl. Conf. on Universal Access in Human-Computer Interaction (UAHCI), Las Vegas, Nevada, USA, July 2005.

[36] M.L. Cascia, S. Sclaroff, V. Athitsos: Fast, reliable head tracking under varying illumination: an approach based on registration of texture– mapped 3D models. IEEE Transactions PAMI, Vol. 22, No. 4, pp. 322–336, 2000.

[37] S. Akyol, J. Zieren: Evaluation of ASM Head Tracker for Robustness against Occlusion. In Proceedings of the International Conference on Imaging Science, Systems, and Technology (CISST 02), Vol. 1, June 2002.

[38] Dongxu Li, Cristian Rodriguez, Xin Yu, and Hongdong Li: Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. In WACV, pages 1459–1469, 2020.

[39] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, Q. Yuan, and A. Thangali. The american sign language lexicon video dataset. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[40] R. Wilbur and A. C. Kak. Purdue rvl-slll american sign language database. 2006.

[41] M. Zahedi, D. Keysers, T. Deselaers, and H. Ney. Combination of tangent distance and an image distortion model for appearance-based sign language recognition. In *Joint Pattern Recognition Symposium*, pages 401–408. Springer, 2005.

[42] K. Grobel and M. Assan. Isolated sign language recognition using hidden markov models. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 1, pages 162–167. IEEE, 1997.

[43] V. S. Kulkarni and S. Lokhande. Appearance based recognition of american sign language using gesture segmentation. *International Journal on Computer Science and Engineering*, 2(03):560–565, 2010.

[44] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti. American sign language recognition with the kinect. In Proceedings of the 13th international conference on multimodal interfaces, pages 279–286. ACM, 2011.

[45] J. Huang,W. Zhou, H. Li, andW. Li. Sign language recognition using 3d convolutional neural networks. In *2015 IEEE international conference on multimedia and expo (ICME)*, pages 1–6. IEEE, 2015.

[46] L. Pigou, M. Van Herreweghe, and J. Dambre. Gesture and sign language recognition with temporal residual networks. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[47] K. M. Lim, A. W. Tan, and S. C. Tan. Block-based histogram of optical flow for isolated sign language recognition. *Journal of Visual Communication and Image Representation*, 40:538–545, 2016.

[48] D. Metaxas, M. Dilsizian, and C. Neidle. Scalable asl sign recognition using model-based machine learning and linguistically annotated corpora. In *8th Workshop on the Representation & Processing of Sign Languages: Involving the Language Community, Language Resources and Evaluation Conference 2018*, 2018.

[49] Q. Xue, X. Li, D. Wang, and W. Zhang. Deep forest-based monocular visual sign language recognition. *Applied Sciences*, 9(9):1945, 2019.

[50] Q. Yang. Chinese sign language recognition based on video sequence appearance modeling. In *2010 5th IEEE Conference on Industrial Electronics and Applications*, pages 1537–1542. IEEE, 2010.

[51] F. Yasir, P. C. Prasad, A. Alsadoon, and A. Elchouemi. Sift based approach on bangla sign language recognition. In *2015 IEEE 8th International Workshop on Computational Intelligence and Applications (IWCIA)*, pages 35–39. IEEE, 2015.

[52] A. Tharwat, T. Gaber, A. E. Hassanien, M. K. Shahin, and B. Refaat. Sift-based arabic sign language recognition system. In *Afro-european conference for industrial advancement*, pages 359–370. Springer, 2015.

[53] S. Liwicki and M. Everingham. Automatic recognition of fingerspelled words in british sign language. In *2009 IEEE computer society conference on computer vision and pattern recognition workshops*, pages 50–57. IEEE, 2009.

[54] P. Buehler, A. Zisserman, and M. Everingham. Learning sign language by watching tv (using weakly aligned subtitles). In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2961–2968. IEEE, 2009.

[55] H. Cooper, E.-J. Ong, N. Pugeault, and R. Bowden. Sign language recognition using sub-units. *Journal of Machine Learning Research*, 13(Jul):2205–2231, 2012.

[56] M. Al-Rousan, K. Assaleh, and A. Talaa. Video-based signer-independent arabic sign language recognition using hidden markov models. *Applied Soft Computing*, 9(3):990–999, 2009.

[57] P. C. Badhe and V. Kulkarni. Indian sign language translator using gesture recognition algorithm. In *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*, pages 195–200. IEEE, 2015.

[58] T. E. Starner. Visual recognition of american sign language using hidden markov models. Technical report, Massachusetts Inst Of Tech Cambridge Dept Of Brain And Cognitive Sciences, 1995.

[59] T. Starner, J. Weaver, and A. Pentland. Real-time American sign language recognition using desk and wearable computer based video. *IEEE Transactions on pattern analysis and machine intelligence*, 20(12):1371–1375, 1998.

[60] J. F. Lichtenauer, E. A. Hendriks, and M. J. Reinders. Sign language recognition by combining statistical dtw and independent classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):2040–2046, 2008.

[61] S. Nagarajan and T. Subashini. Static hand gesture recognition for sign language alphabets using edge oriented histogram and multi class svm. *International Journal of Computer Applications*, 82(4), 2013.

[62] H. Shin, W. J. Kim, and K.-a. Jang. Korean sign language recognition based on image and convolution neural network. In *Proceedings of the 2nd International Conference on Image and Graphics Processing*, pages 52–55. ACM, 2019.

[63] P. Kishore, G. A. Rao, E. K. Kumar, M. T. K. Kumar, and D. A. Kumar. Selfie sign language recognition with convolutional neural networks. *International Journal of Intelligent Systems and Applications*, 10(10):63, 2018.

[64] S.-K. Ko, J. G. Son, and H. Jung. Sign language recognition with recurrent neural network using human keypoint detection. In *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, pages 326–328. ACM, 2018.

[65] S.-K. Ko, C. J. Kim, H. Jung, and C. Cho. Neural sign language translation based on human keypoint estimation. *Applied Sciences*, 9(13):2683, 2019.

[66] Y. Ye, Y. Tian, M. Huenerfauth, and J. Liu. Recognizing american sign language gestures from within continuous videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2064–2073, 2018.

[67] Asl university. http://asluniversity.com/.

[68] N. K. Caselli, Z. S. Sehyr, A. M. Cohen-Goldberg, and K. Emmorey. Asl-lex: A lexical database of american sign language. Behavior research methods, 49(2):784–801, 2017.

[69] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

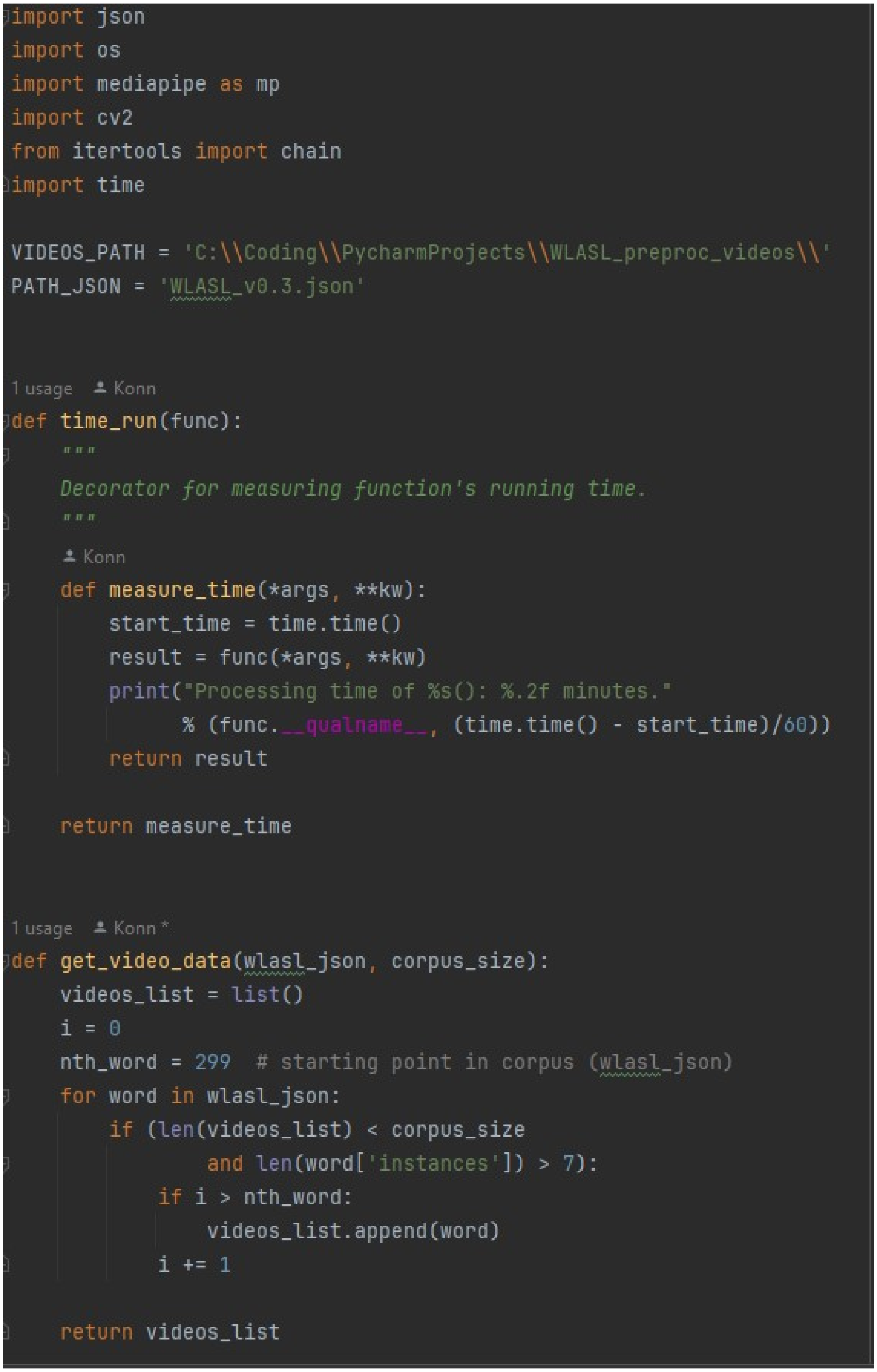
[70] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

[71] C. McCaskill, C. Lucas, R. Bayley, and J. Hill. The hidden treasure of Black ASL: Its history and structure. Gallaudet University Press Washington, DC, 2011.

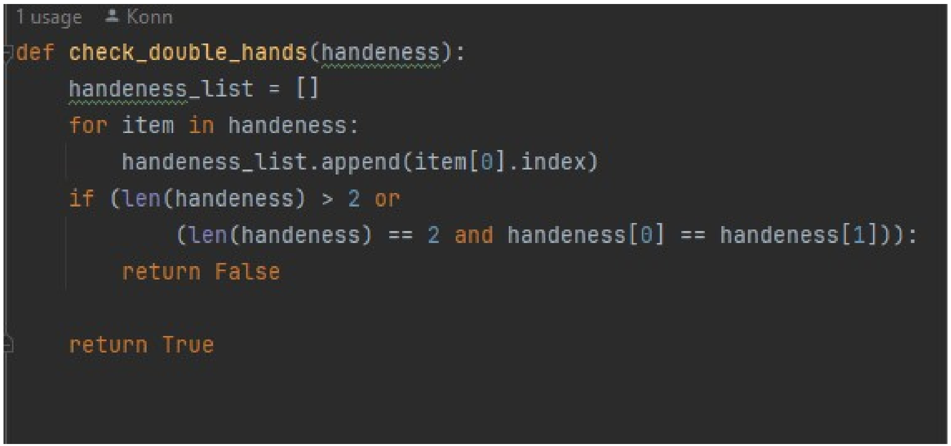
[72] Loffe, S., Szegedy (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456).

Appendix

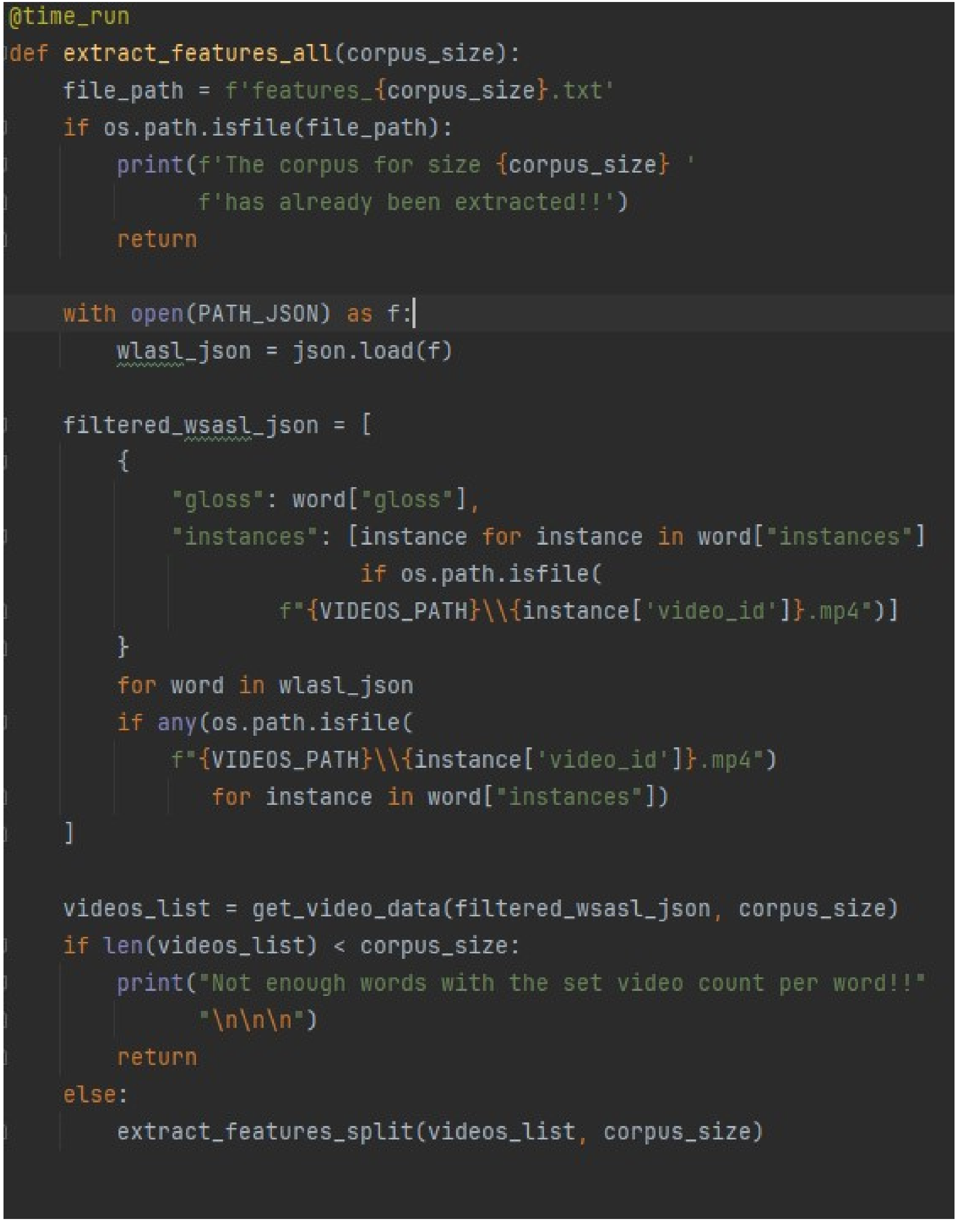
Code for extracting feature landmarks from the available videos.



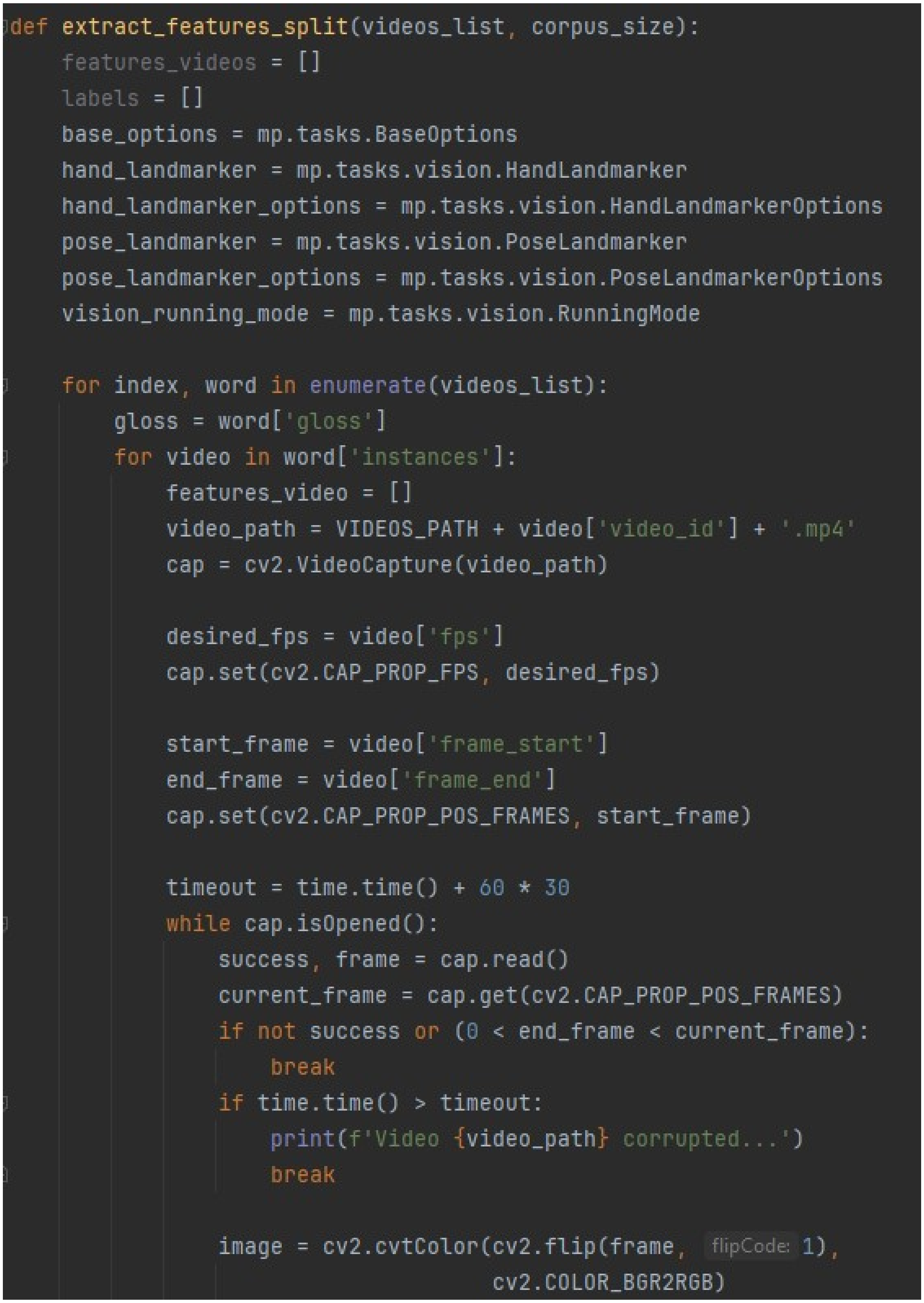
```python
import json
import os
import mediapipe as mp
import cv2
from itertools import chain
import time

VIDEOS_PATH = 'C:\\Coding\\PycharmProjects\\WLASL_preproc_videos\\'
PATH_JSON = 'WLASL_v0.3.json'


def time_run(func):
    """
    Decorator for measuring function's running time.
    """
    def measure_time(*args, **kw):
        start_time = time.time()
        result = func(*args, **kw)
        print("Processing time of %s(): %.2f minutes."
              % (func.__qualname__, (time.time() - start_time)/60))
        return result

    return measure_time


def get_video_data(wlasl_json, corpus_size):
    videos_list = list()
    i = 0
    nth_word = 299  # starting point in corpus (wlasl_json)
    for word in wlasl_json:
        if (len(videos_list) < corpus_size
                and len(word['instances']) > 7):
            if i > nth_word:
                videos_list.append(word)
            i += 1

    return videos_list
```



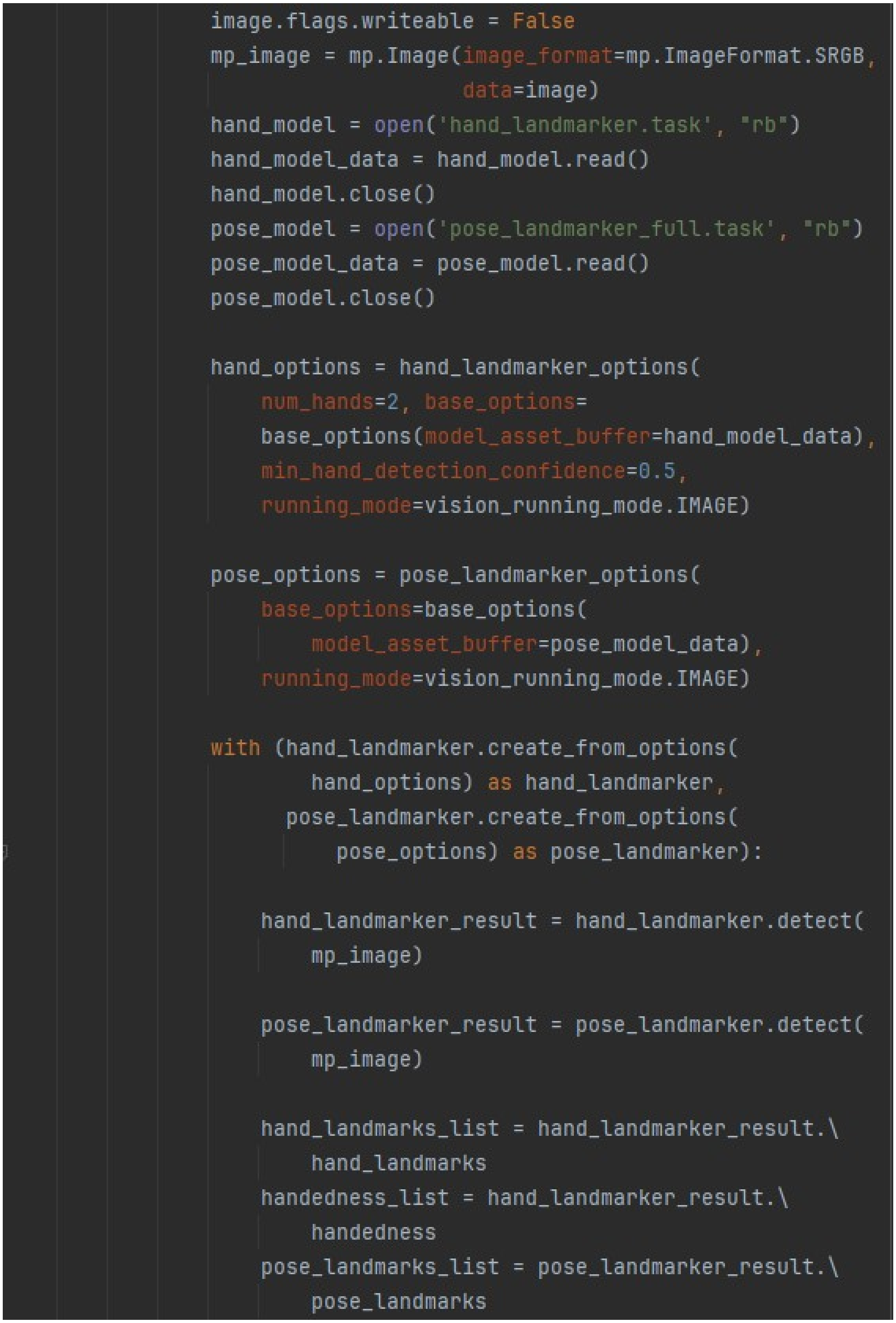
```python
def check_double_hands(handeness):
    handeness_list = []
    for item in handeness:
        handeness_list.append(item[0].index)
    if (len(handeness) > 2 or
            (len(handeness) == 2 and handeness[0] == handeness[1])):
        return False

    return True
```



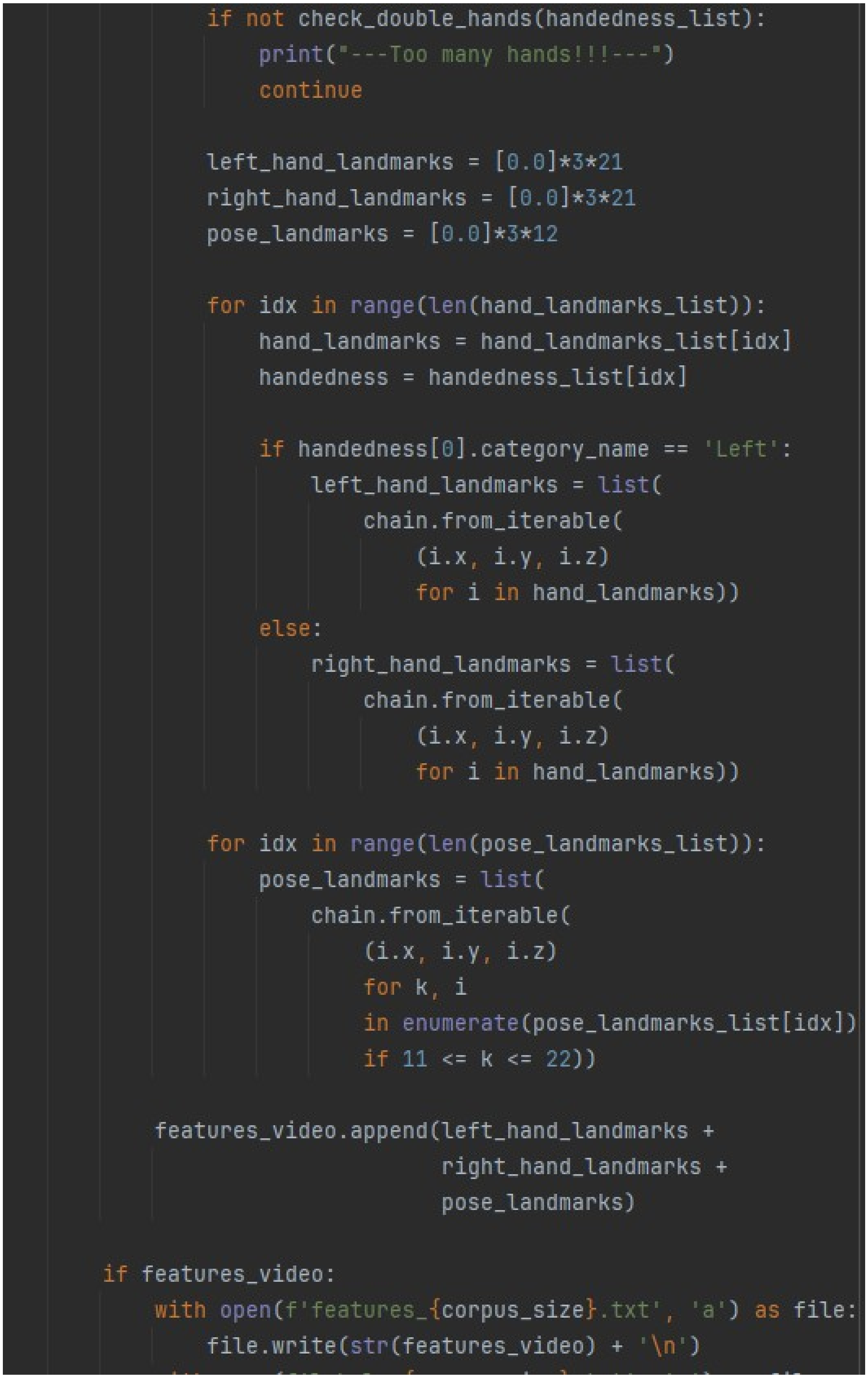
```python
@time_run
def extract_features_all(corpus_size):
    file_path = f'features_{corpus_size}.txt'
    if os.path.isfile(file_path):
        print(f'The corpus for size {corpus_size} '
              f'has already been extracted!!')
        return

    with open(PATH_JSON) as f:
        wlasl_json = json.load(f)

    filtered_wsasl_json = [
        {
            "gloss": word["gloss"],
            "instances": [instance for instance in word["instances"]
                          if os.path.isfile(
                    f"{VIDEOS_PATH}\\{instance['video_id']}.mp4")]
        }
        for word in wlasl_json
        if any(os.path.isfile(
            f"{VIDEOS_PATH}\\{instance['video_id']}.mp4")
               for instance in word["instances"])
    ]

    videos_list = get_video_data(filtered_wsasl_json, corpus_size)
    if len(videos_list) < corpus_size:
        print("Not enough words with the set video count per word!!"
              "\n\n\n")
        return
    else:
        extract_features_split(videos_list, corpus_size)
```



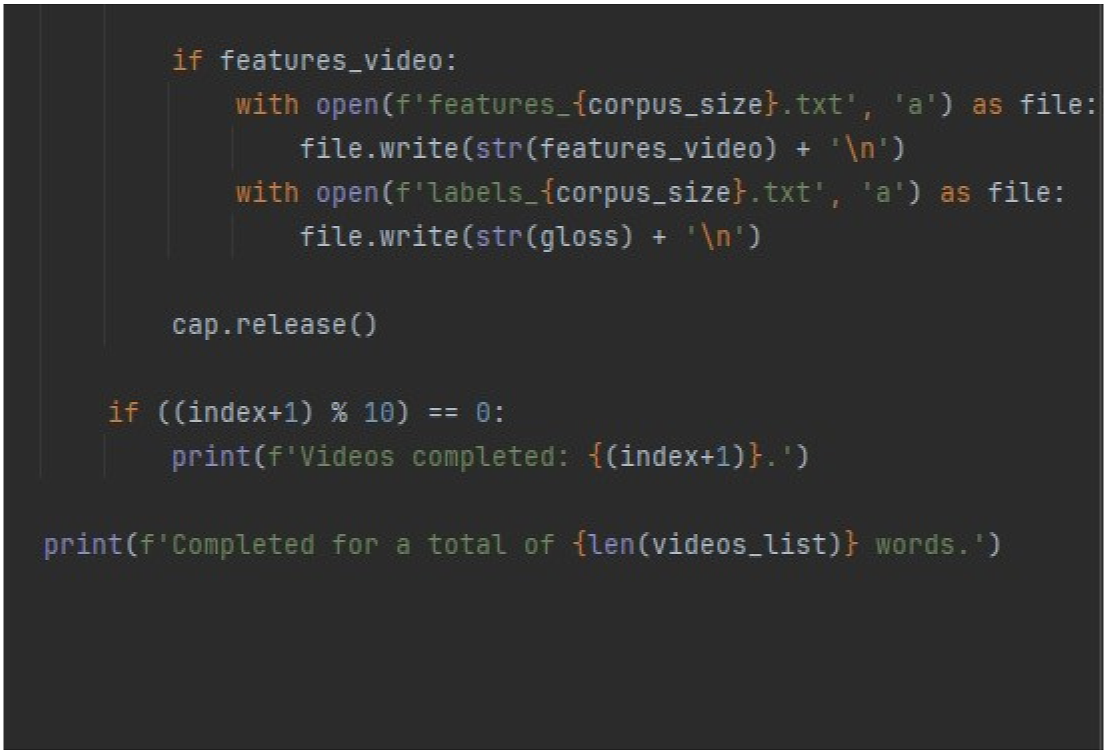
```python
def extract_features_split(videos_list, corpus_size):
    features_videos = []
    labels = []
    base_options = mp.tasks.BaseOptions
    hand_landmarker = mp.tasks.vision.HandLandmarker
    hand_landmarker_options = mp.tasks.vision.HandLandmarkerOptions
    pose_landmarker = mp.tasks.vision.PoseLandmarker
    pose_landmarker_options = mp.tasks.vision.PoseLandmarkerOptions
    vision_running_mode = mp.tasks.vision.RunningMode

    for index, word in enumerate(videos_list):
        gloss = word['gloss']
        for video in word['instances']:
            features_video = []
            video_path = VIDEOS_PATH + video['video_id'] + '.mp4'
            cap = cv2.VideoCapture(video_path)

            desired_fps = video['fps']
            cap.set(cv2.CAP_PROP_FPS, desired_fps)

            start_frame = video['frame_start']
            end_frame = video['frame_end']
            cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

            timeout = time.time() + 60 * 30
            while cap.isOpened():
                success, frame = cap.read()
                current_frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
                if not success or (0 < end_frame < current_frame):
                    break
                if time.time() > timeout:
                    print(f'Video {video_path} corrupted...')
                    break

                image = cv2.cvtColor(cv2.flip(frame, flipCode: 1),
                                     cv2.COLOR_BGR2RGB)
```



```
                image.flags.writeable = False
                mp_image = mp.Image(image_format=mp.ImageFormat.SRGB,
                                    data=image)
                hand_model = open('hand_landmarker.task', "rb")
                hand_model_data = hand_model.read()
                hand_model.close()
                pose_model = open('pose_landmarker_full.task', "rb")
                pose_model_data = pose_model.read()
                pose_model.close()

                hand_options = hand_landmarker_options(
                    num_hands=2, base_options=
                    base_options(model_asset_buffer=hand_model_data),
                    min_hand_detection_confidence=0.5,
                    running_mode=vision_running_mode.IMAGE)

                pose_options = pose_landmarker_options(
                    base_options=base_options(
                        model_asset_buffer=pose_model_data),
                    running_mode=vision_running_mode.IMAGE)

                with (hand_landmarker.create_from_options(
                        hand_options) as hand_landmarker,
                      pose_landmarker.create_from_options(
                          pose_options) as pose_landmarker):

                    hand_landmarker_result = hand_landmarker.detect(
                        mp_image)

                    pose_landmarker_result = pose_landmarker.detect(
                        mp_image)

                    hand_landmarks_list = hand_landmarker_result.\
                        hand_landmarks
                    handedness_list = hand_landmarker_result.\
                        handedness
                    pose_landmarks_list = pose_landmarker_result.\
                        pose_landmarks
```

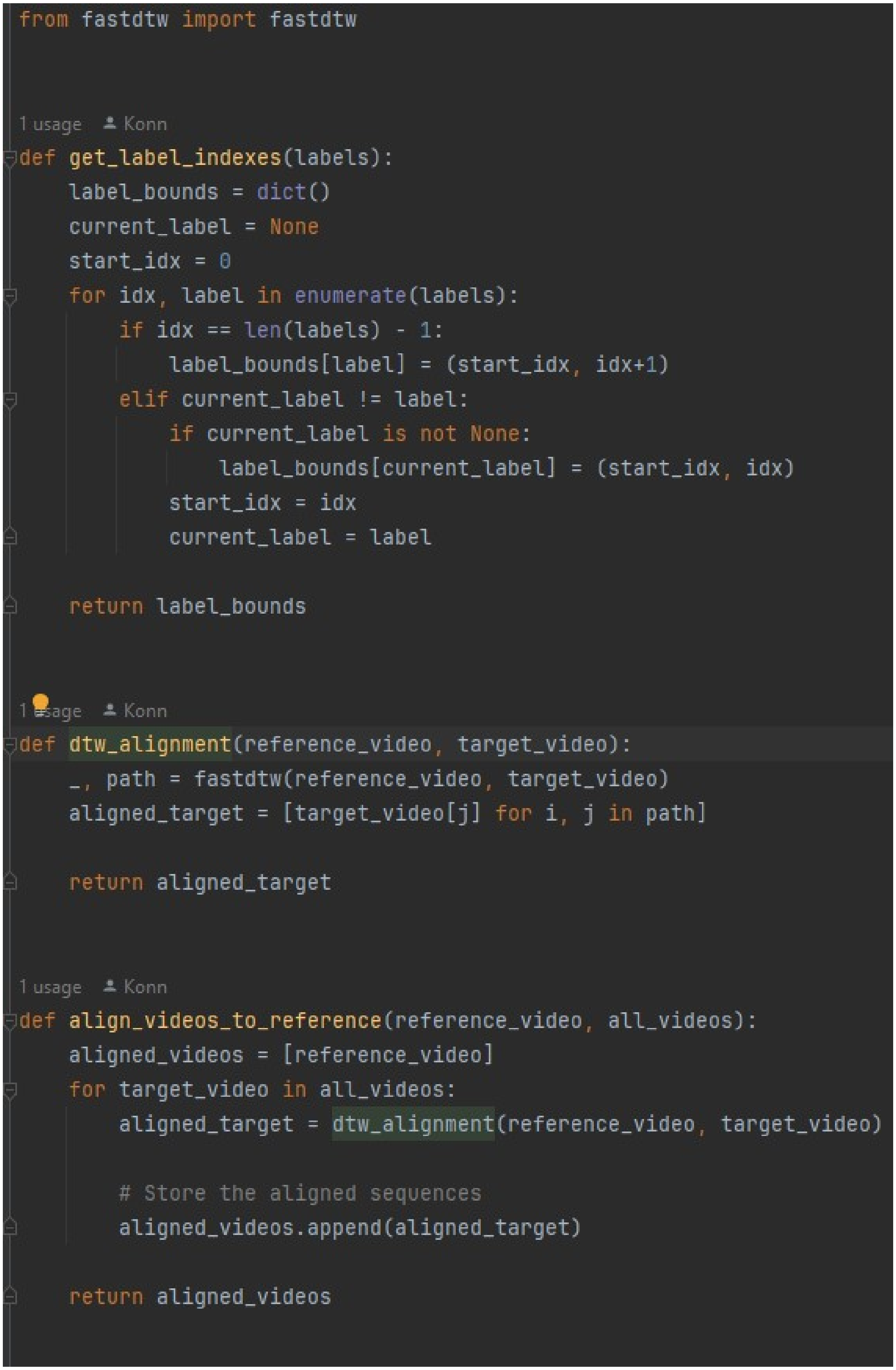


```
                if not check_double_hands(handedness_list):
                    print("---Too many hands!!!---")
                    continue

                left_hand_landmarks = [0.0]*3*21
                right_hand_landmarks = [0.0]*3*21
                pose_landmarks = [0.0]*3*12

                for idx in range(len(hand_landmarks_list)):
                    hand_landmarks = hand_landmarks_list[idx]
                    handedness = handedness_list[idx]

                    if handedness[0].category_name == 'Left':
                        left_hand_landmarks = list(
                            chain.from_iterable(
                                (i.x, i.y, i.z)
                                for i in hand_landmarks))
                    else:
                        right_hand_landmarks = list(
                            chain.from_iterable(
                                (i.x, i.y, i.z)
                                for i in hand_landmarks))

                for idx in range(len(pose_landmarks_list)):
                    pose_landmarks = list(
                        chain.from_iterable(
                            (i.x, i.y, i.z)
                            for k, i
                            in enumerate(pose_landmarks_list[idx])
                            if 11 <= k <= 22))

            features_video.append(left_hand_landmarks +
                                  right_hand_landmarks +
                                  pose_landmarks)

        if features_video:
            with open(f'features_{corpus_size}.txt', 'a') as file:
                file.write(str(features_video) + '\n')
```

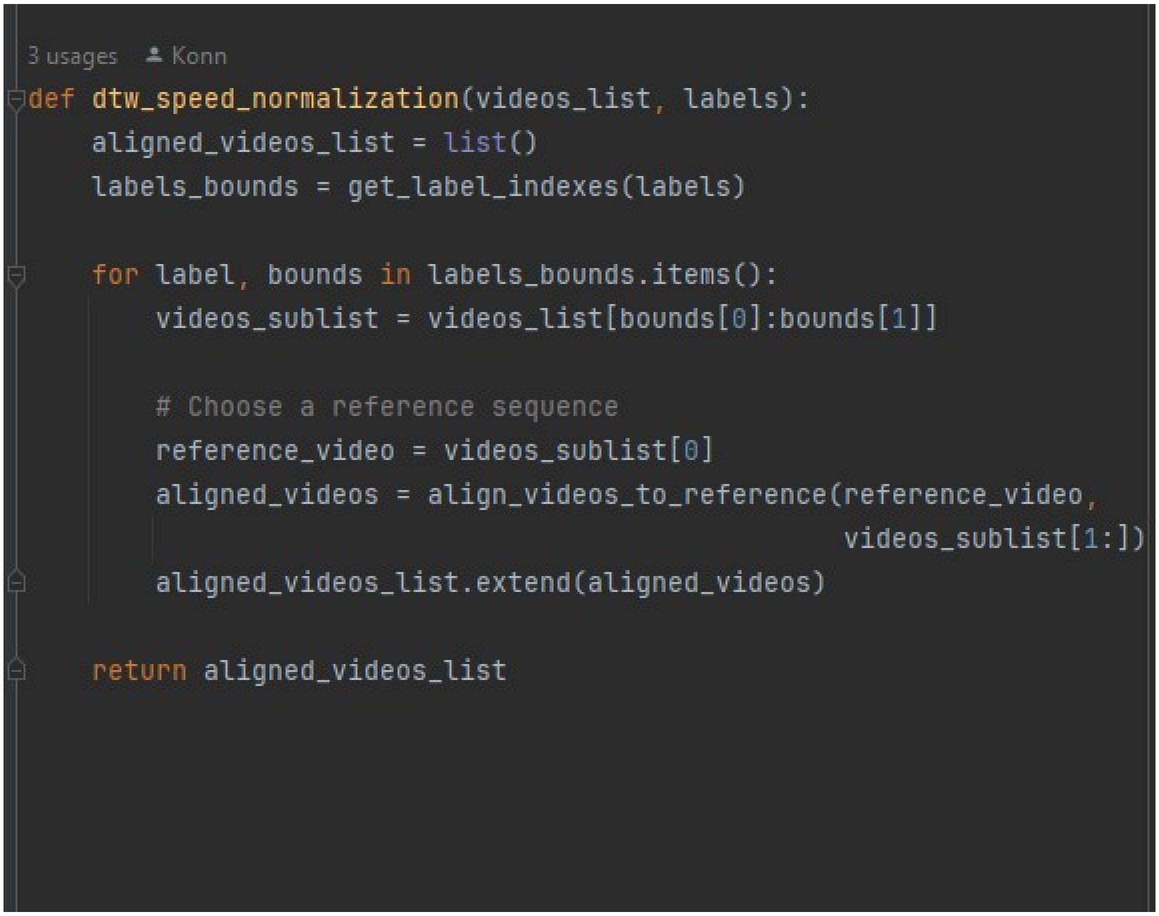


```
        if features_video:
            with open(f'features_{corpus_size}.txt', 'a') as file:
                file.write(str(features_video) + '\n')
            with open(f'labels_{corpus_size}.txt', 'a') as file:
                file.write(str(gloss) + '\n')

        cap.release()

    if ((index+1) % 10) == 0:
        print(f'Videos completed: {(index+1)}.')

print(f'Completed for a total of {len(videos_list)} words.')
```

Code for Dynamic Time Warping.

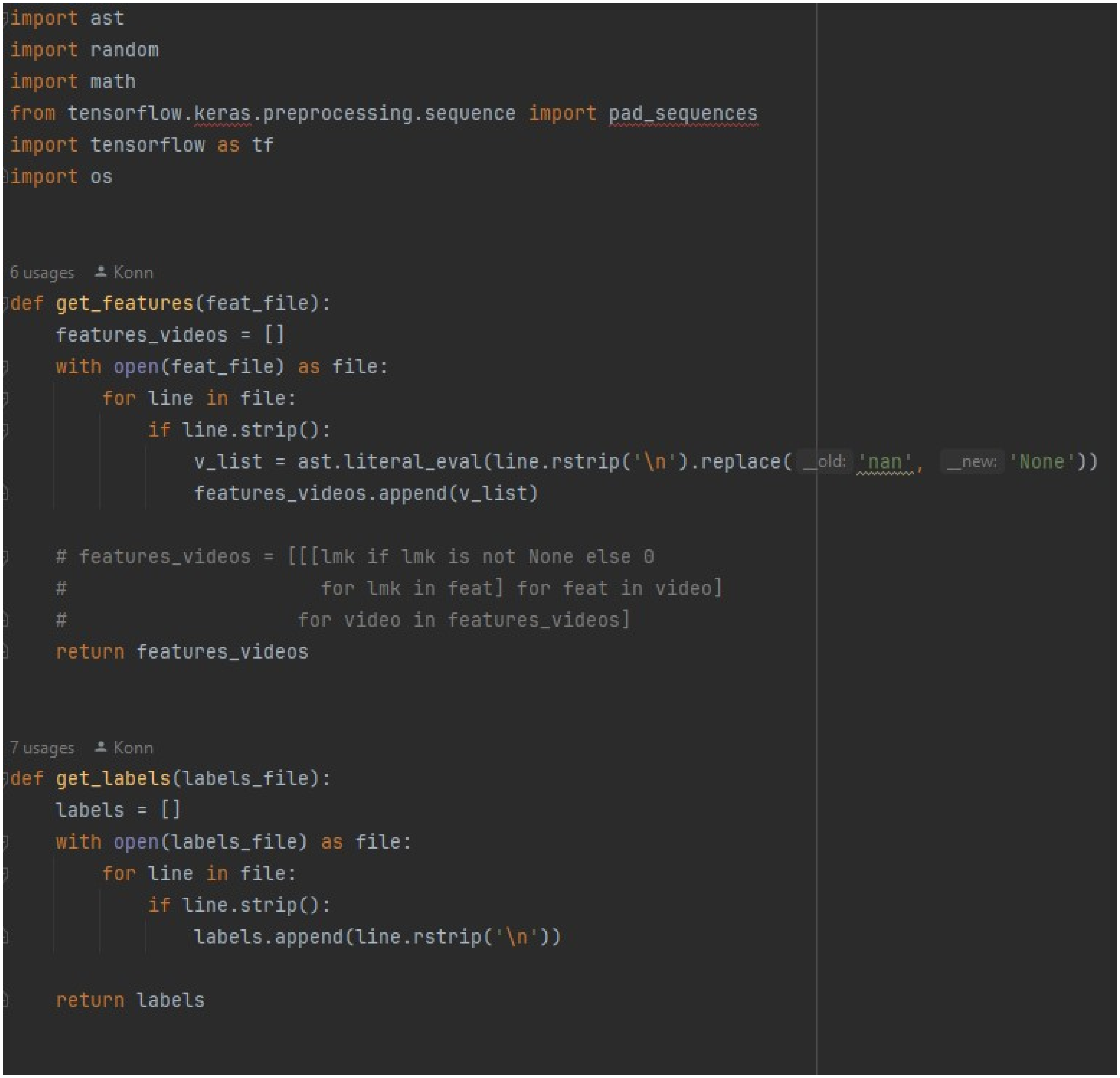


```python
from fastdtw import fastdtw


def get_label_indexes(labels):
    label_bounds = dict()
    current_label = None
    start_idx = 0
    for idx, label in enumerate(labels):
        if idx == len(labels) - 1:
            label_bounds[label] = (start_idx, idx+1)
        elif current_label != label:
            if current_label is not None:
                label_bounds[current_label] = (start_idx, idx)
            start_idx = idx
            current_label = label

    return label_bounds


def dtw_alignment(reference_video, target_video):
    _, path = fastdtw(reference_video, target_video)
    aligned_target = [target_video[j] for i, j in path]

    return aligned_target


def align_videos_to_reference(reference_video, all_videos):
    aligned_videos = [reference_video]
    for target_video in all_videos:
        aligned_target = dtw_alignment(reference_video, target_video)

        # Store the aligned sequences
        aligned_videos.append(aligned_target)

    return aligned_videos
```

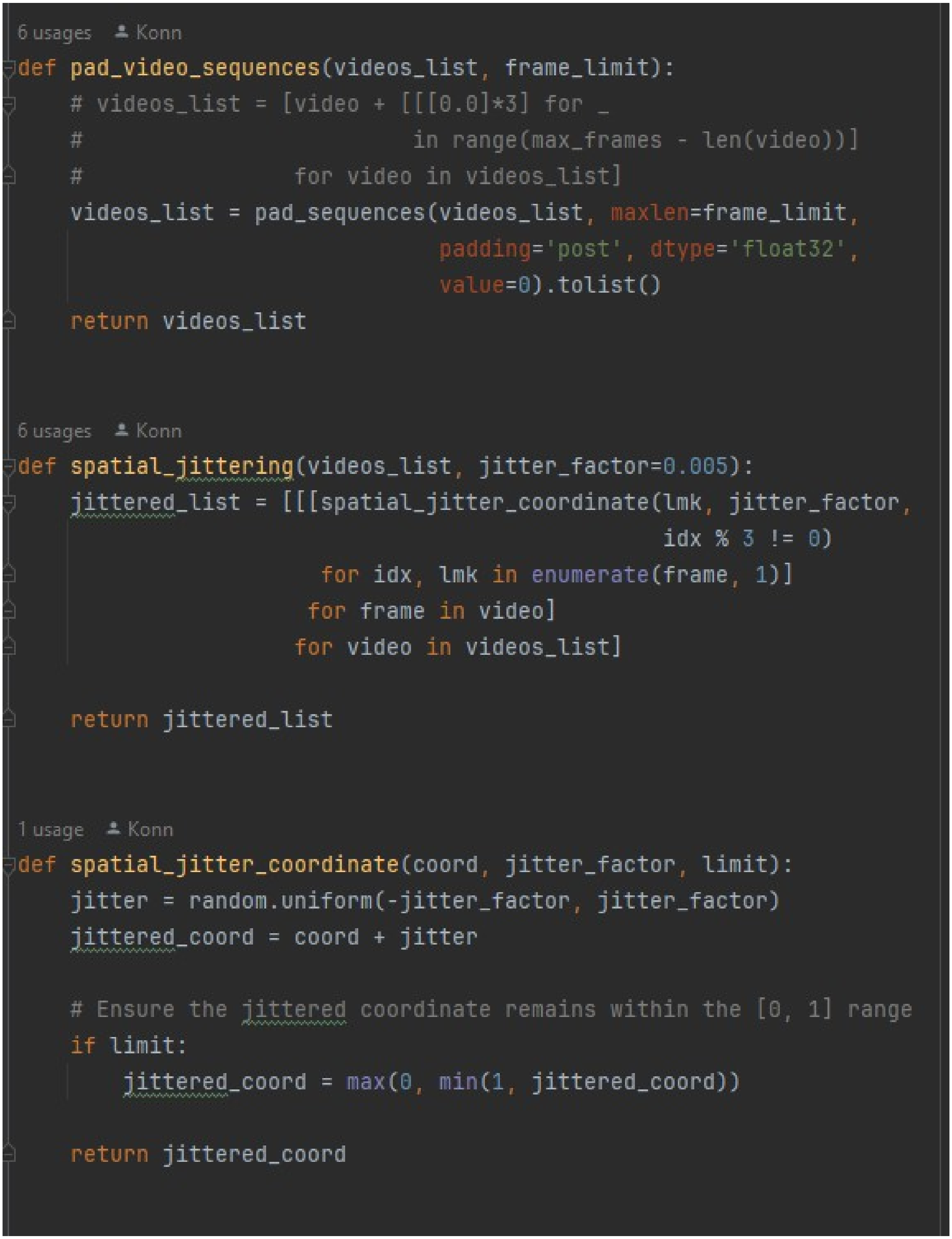


```python
def dtw_speed_normalization(videos_list, labels):
    aligned_videos_list = list()
    labels_bounds = get_label_indexes(labels)

    for label, bounds in labels_bounds.items():
        videos_sublist = videos_list[bounds[0]:bounds[1]]

        # Choose a reference sequence
        reference_video = videos_sublist[0]
        aligned_videos = align_videos_to_reference(reference_video,
                                                   videos_sublist[1:])
        aligned_videos_list.extend(aligned_videos)

    return aligned_videos_list
```

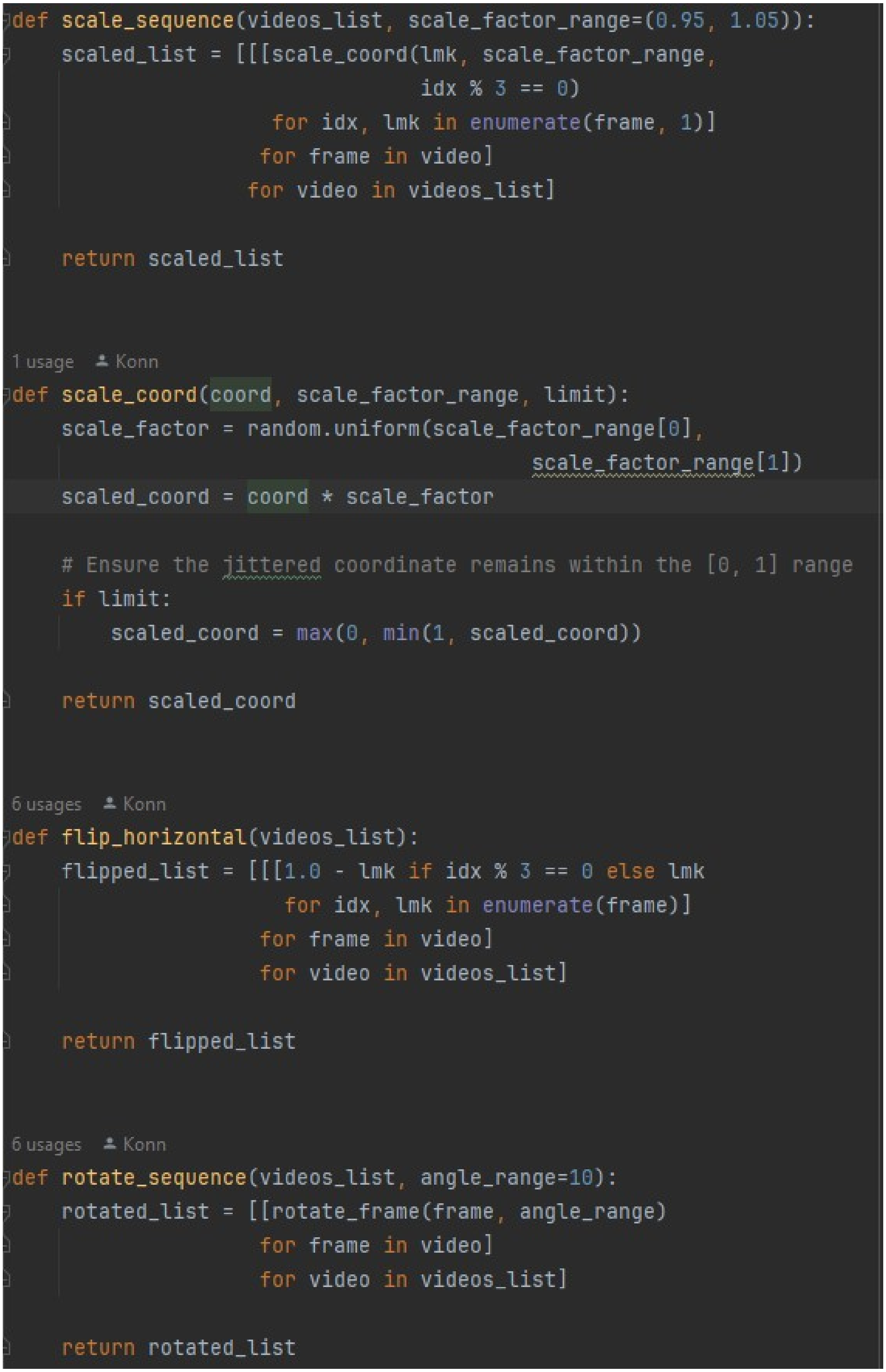
Code for various data utilities like landmark transformation or sequences padding.



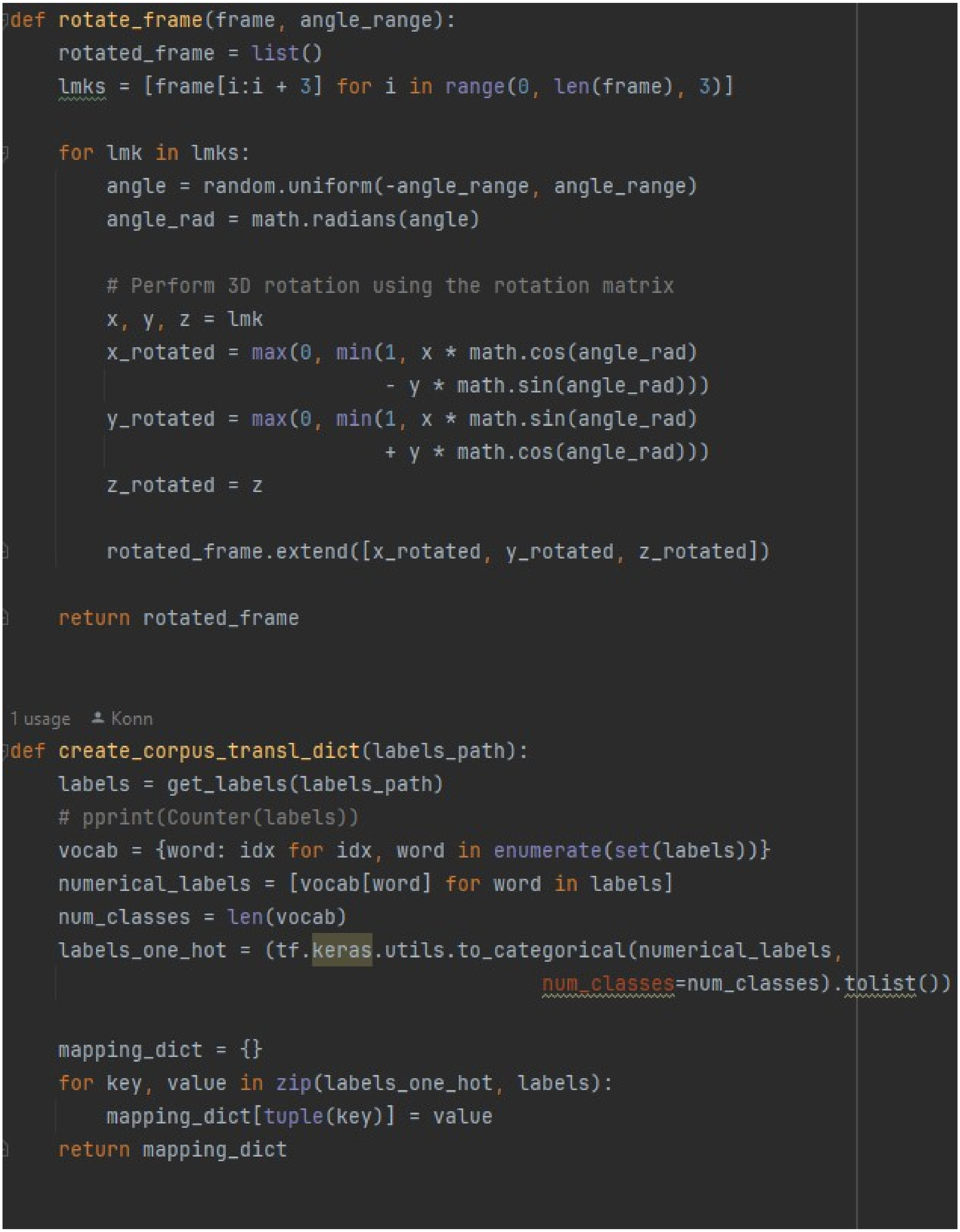
```
import ast
import random
import math
from tensorflow.keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
import os


6 usages  Konn
def get_features(feat_file):
    features_videos = []
    with open(feat_file) as file:
        for line in file:
            if line.strip():
                v_list = ast.literal_eval(line.rstrip('\n').replace( __old: 'nan',  __new: 'None'))
                features_videos.append(v_list)

    # features_videos = [[[lmk if lmk is not None else 0
    #                       for lmk in feat] for feat in video]
    #                      for video in features_videos]
    return features_videos


7 usages  Konn
def get_labels(labels_file):
    labels = []
    with open(labels_file) as file:
        for line in file:
            if line.strip():
                labels.append(line.rstrip('\n'))

    return labels
```



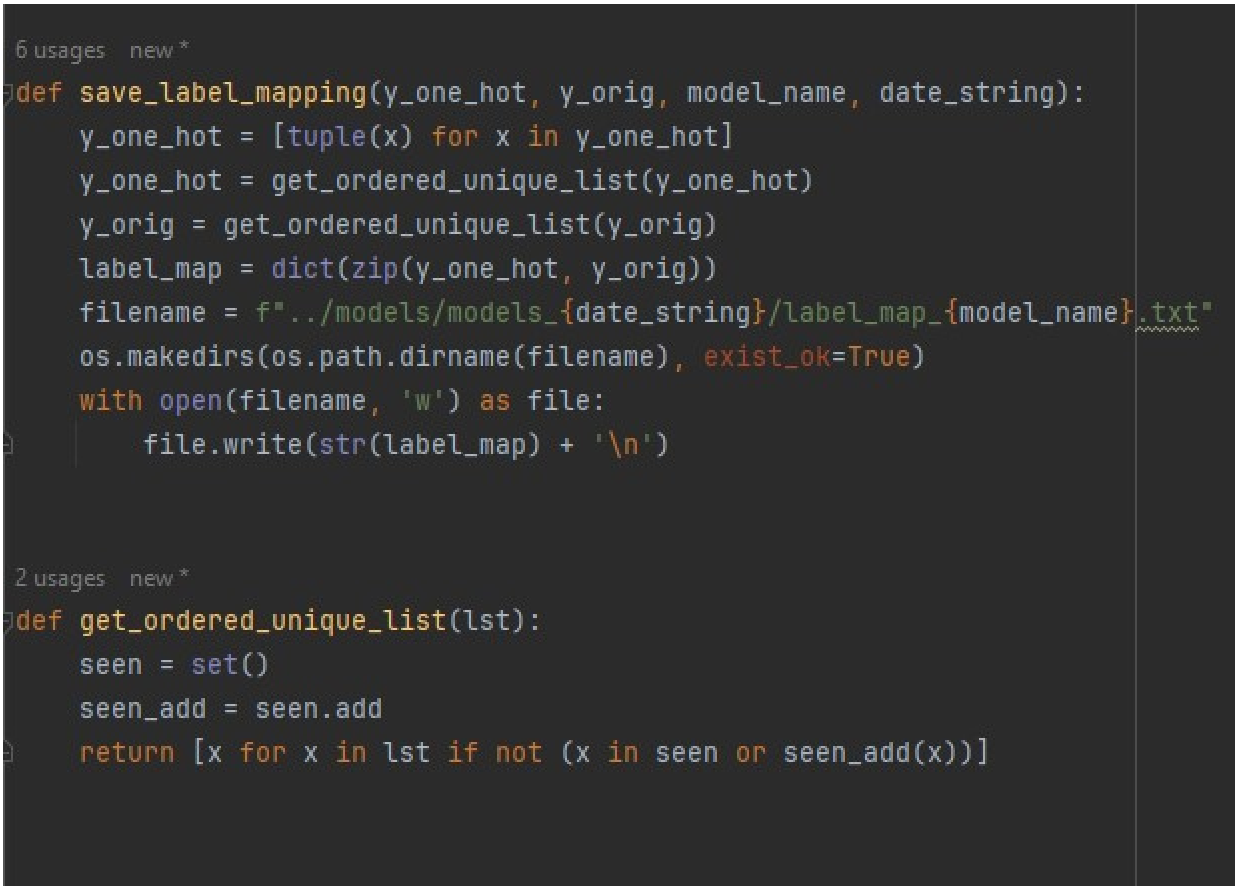
```
6 usages  ≗ Konn
def pad_video_sequences(videos_list, frame_limit):
    # videos_list = [video + [[[0.0]*3] for _
    #                          in range(max_frames - len(video))]
    #                 for video in videos_list]
    videos_list = pad_sequences(videos_list, maxlen=frame_limit,
                                padding='post', dtype='float32',
                                value=0).tolist()
    return videos_list


6 usages  ≗ Konn
def spatial_jittering(videos_list, jitter_factor=0.005):
    jittered_list = [[[spatial_jitter_coordinate(lmk, jitter_factor,
                                                 idx % 3 != 0)
                       for idx, lmk in enumerate(frame, 1)]
                      for frame in video]
                     for video in videos_list]

    return jittered_list


1 usage  ≗ Konn
def spatial_jitter_coordinate(coord, jitter_factor, limit):
    jitter = random.uniform(-jitter_factor, jitter_factor)
    jittered_coord = coord + jitter

    # Ensure the jittered coordinate remains within the [0, 1] range
    if limit:
        jittered_coord = max(0, min(1, jittered_coord))

    return jittered_coord
```



```
def scale_sequence(videos_list, scale_factor_range=(0.95, 1.05)):
    scaled_list = [[[scale_coord(lmk, scale_factor_range,
                                 idx % 3 == 0)
                     for idx, lmk in enumerate(frame, 1)]
                    for frame in video]
                   for video in videos_list]

    return scaled_list


1 usage  Konn
def scale_coord(coord, scale_factor_range, limit):
    scale_factor = random.uniform(scale_factor_range[0],
                                          scale_factor_range[1])
    scaled_coord = coord * scale_factor

    # Ensure the jittered coordinate remains within the [0, 1] range
    if limit:
        scaled_coord = max(0, min(1, scaled_coord))

    return scaled_coord


6 usages  Konn
def flip_horizontal(videos_list):
    flipped_list = [[[1.0 - lmk if idx % 3 == 0 else lmk
                      for idx, lmk in enumerate(frame)]
                    for frame in video]
                    for video in videos_list]

    return flipped_list


6 usages  Konn
def rotate_sequence(videos_list, angle_range=10):
    rotated_list = [[rotate_frame(frame, angle_range)
                    for frame in video]
                    for video in videos_list]

    return rotated_list
```

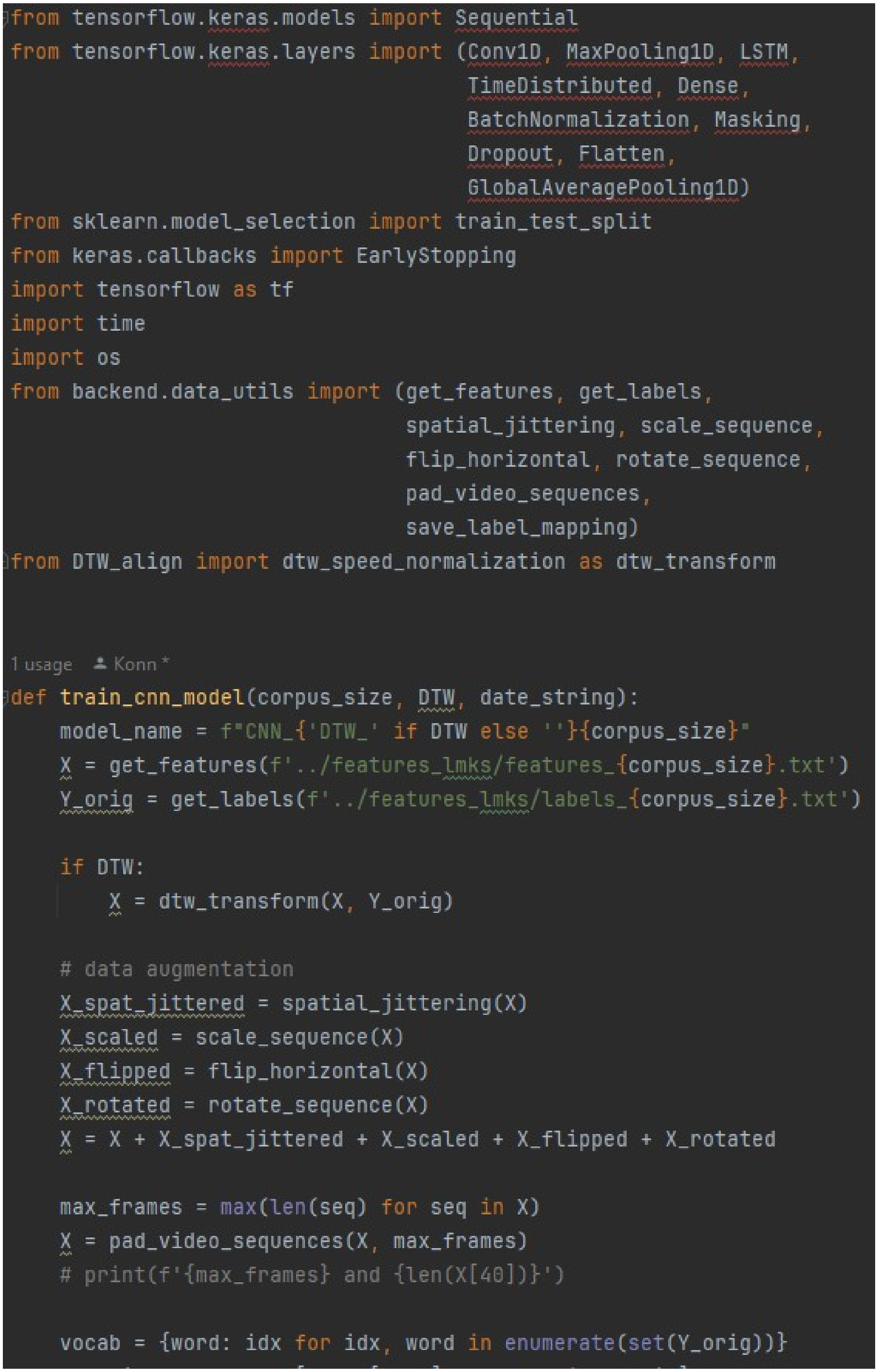


```python
def rotate_frame(frame, angle_range):
    rotated_frame = list()
    lmks = [frame[i:i + 3] for i in range(0, len(frame), 3)]

    for lmk in lmks:
        angle = random.uniform(-angle_range, angle_range)
        angle_rad = math.radians(angle)

        # Perform 3D rotation using the rotation matrix
        x, y, z = lmk
        x_rotated = max(0, min(1, x * math.cos(angle_rad)
                               - y * math.sin(angle_rad)))
        y_rotated = max(0, min(1, x * math.sin(angle_rad)
                               + y * math.cos(angle_rad)))
        z_rotated = z

        rotated_frame.extend([x_rotated, y_rotated, z_rotated])

    return rotated_frame


def create_corpus_transl_dict(labels_path):
    labels = get_labels(labels_path)
    # pprint(Counter(labels))
    vocab = {word: idx for idx, word in enumerate(set(labels))}
    numerical_labels = [vocab[word] for word in labels]
    num_classes = len(vocab)
    labels_one_hot = (tf.keras.utils.to_categorical(numerical_labels,
                                            num_classes=num_classes).tolist())

    mapping_dict = {}
    for key, value in zip(labels_one_hot, labels):
        mapping_dict[tuple(key)] = value
    return mapping_dict
```

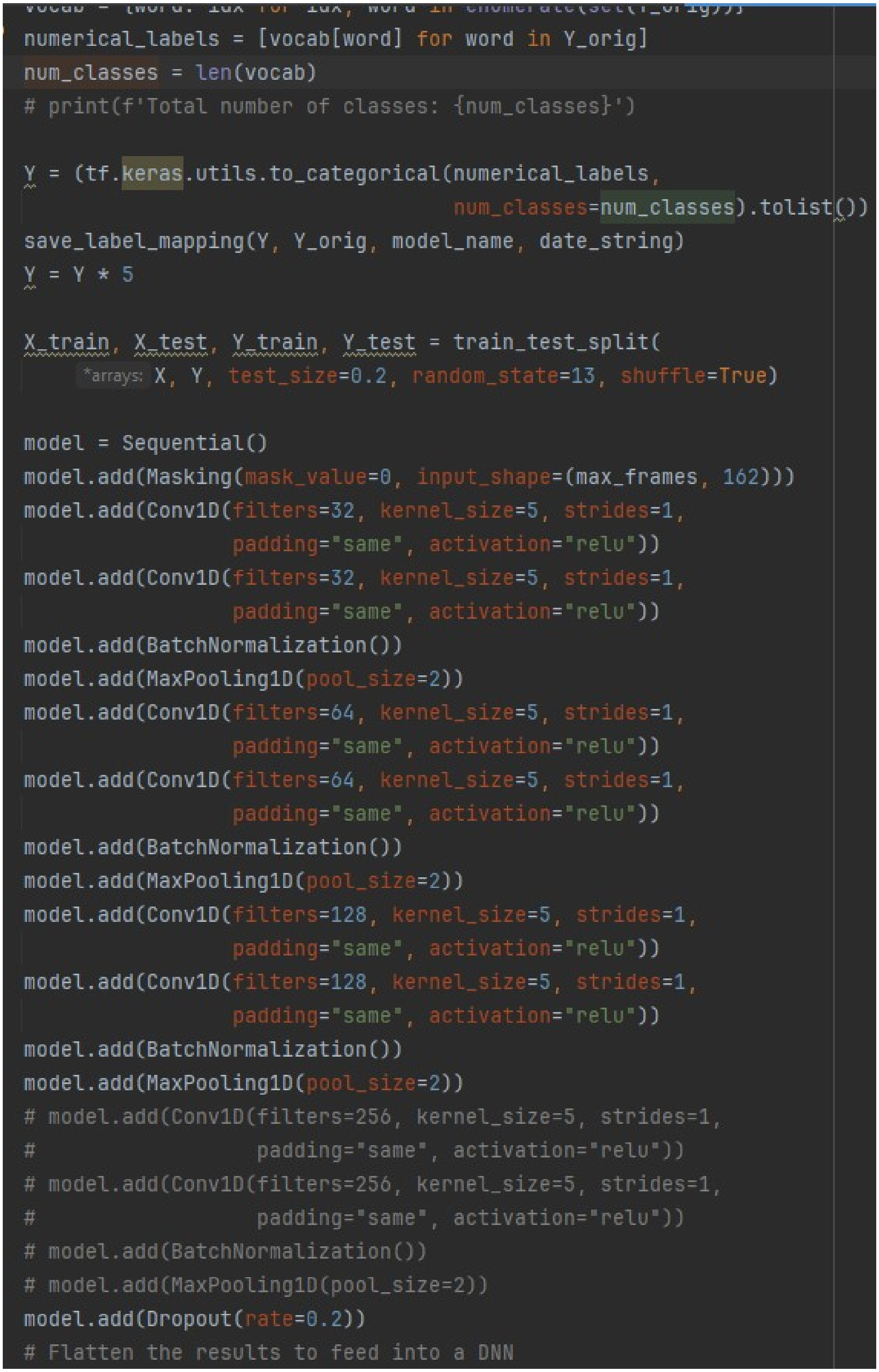


```
def save_label_mapping(y_one_hot, y_orig, model_name, date_string):
    y_one_hot = [tuple(x) for x in y_one_hot]
    y_one_hot = get_ordered_unique_list(y_one_hot)
    y_orig = get_ordered_unique_list(y_orig)
    label_map = dict(zip(y_one_hot, y_orig))
    filename = f"../models/models_{date_string}/label_map_{model_name}.txt"
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with open(filename, 'w') as file:
        file.write(str(label_map) + '\n')


def get_ordered_unique_list(lst):
    seen = set()
    seen_add = seen.add
    return [x for x in lst if not (x in seen or seen_add(x))]
```

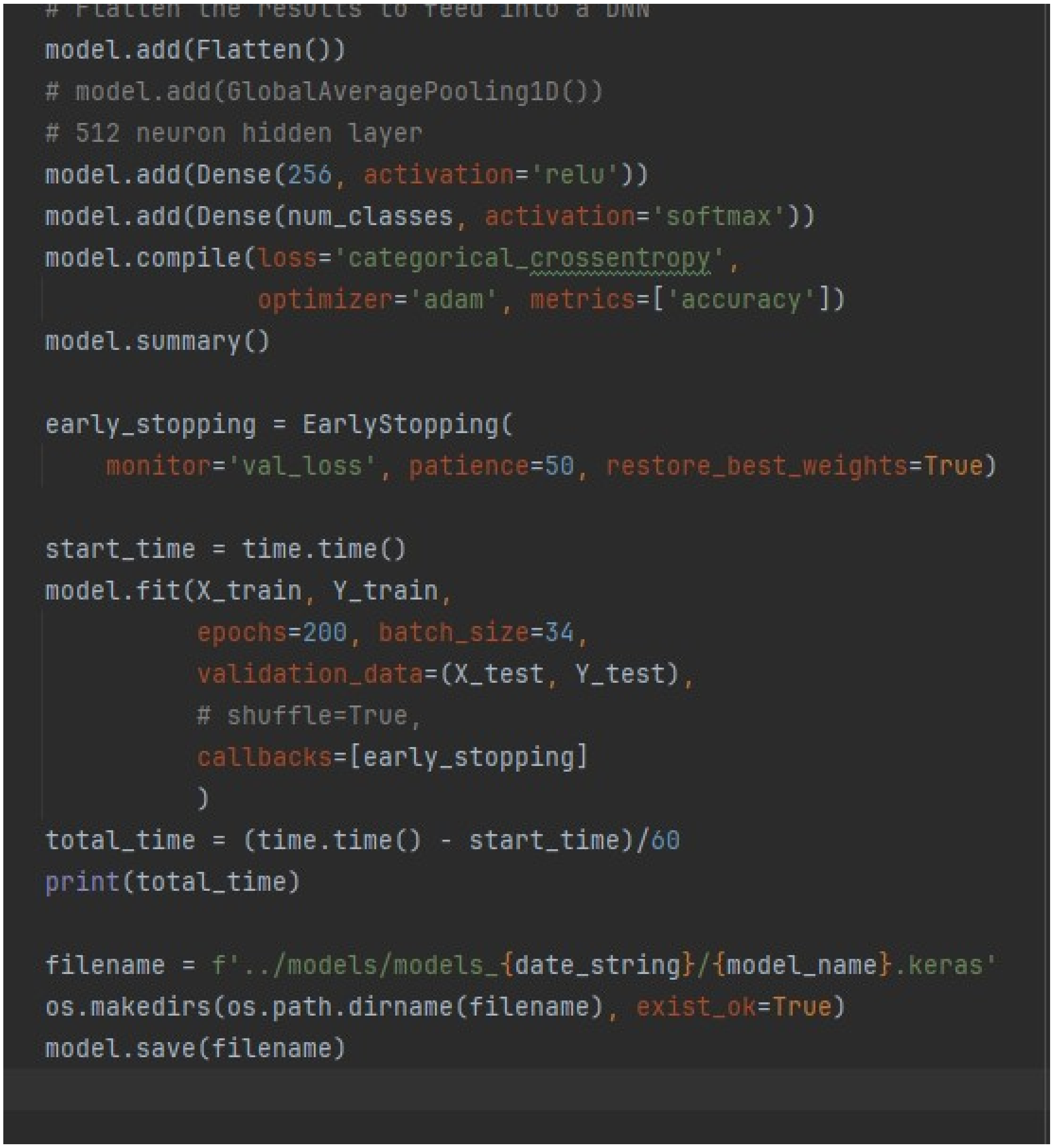
Code for training the CNN model.



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv1D, MaxPooling1D, LSTM,
                                     TimeDistributed, Dense,
                                     BatchNormalization, Masking,
                                     Dropout, Flatten,
                                     GlobalAveragePooling1D)
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
import tensorflow as tf
import time
import os
from backend.data_utils import (get_features, get_labels,
                                spatial_jittering, scale_sequence,
                                flip_horizontal, rotate_sequence,
                                pad_video_sequences,
                                save_label_mapping)
from DTW_align import dtw_speed_normalization as dtw_transform


1 usage  Konn *
def train_cnn_model(corpus_size, DTW, date_string):
    model_name = f"CNN_{'DTW_' if DTW else ''}{corpus_size}"
    X = get_features(f'../features_lmks/features_{corpus_size}.txt')
    Y_orig = get_labels(f'../features_lmks/labels_{corpus_size}.txt')

    if DTW:
        X = dtw_transform(X, Y_orig)

    # data augmentation
    X_spat_jittered = spatial_jittering(X)
    X_scaled = scale_sequence(X)
    X_flipped = flip_horizontal(X)
    X_rotated = rotate_sequence(X)
    X = X + X_spat_jittered + X_scaled + X_flipped + X_rotated

    max_frames = max(len(seq) for seq in X)
    X = pad_video_sequences(X, max_frames)
    # print(f'{max_frames} and {len(X[40])}')

    vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
```



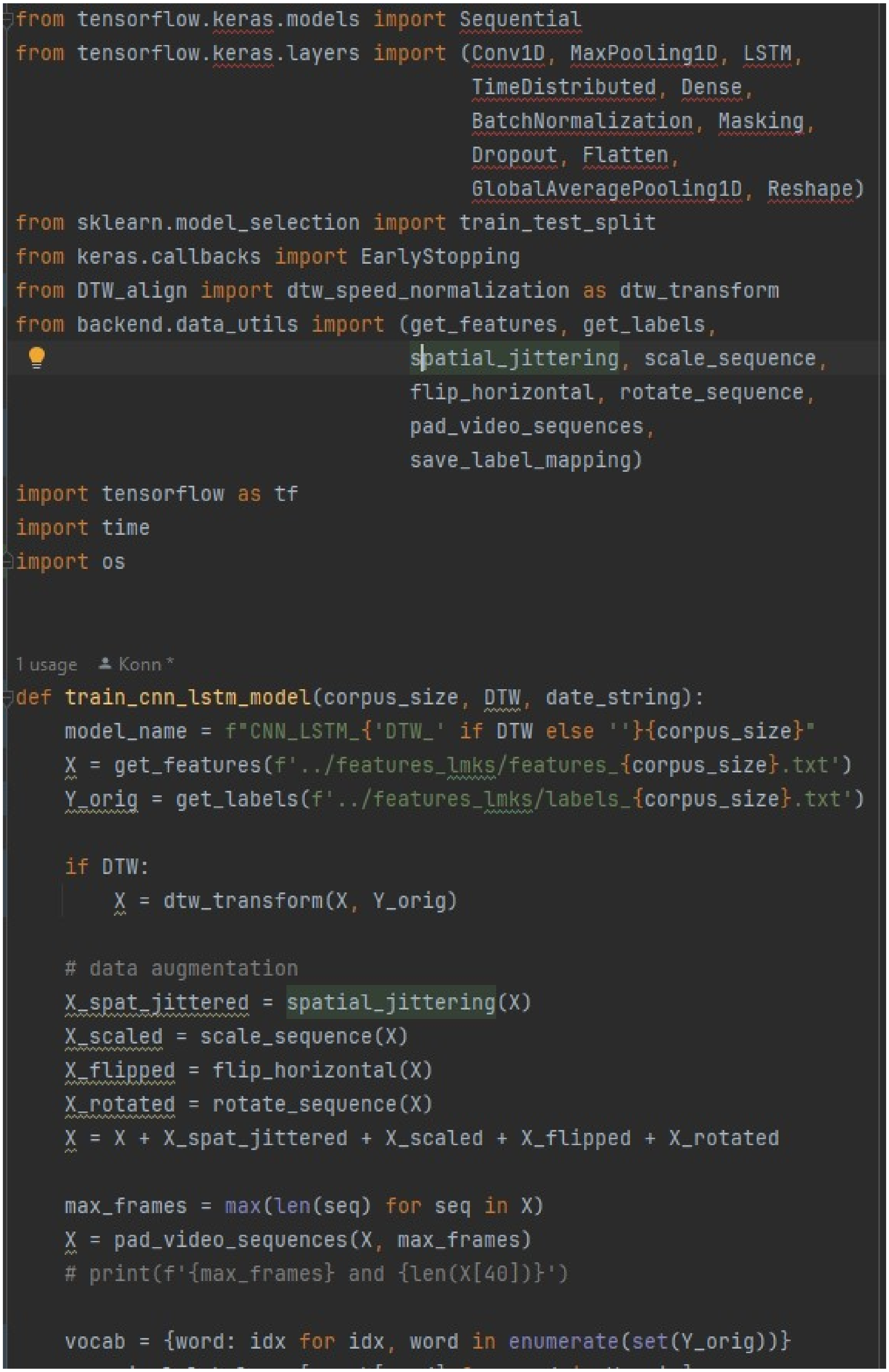
```
vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
numerical_labels = [vocab[word] for word in Y_orig]
num_classes = len(vocab)
# print(f'Total number of classes: {num_classes}')

Y = (tf.keras.utils.to_categorical(numerical_labels,
                                  num_classes=num_classes).tolist())
save_label_mapping(Y, Y_orig, model_name, date_string)
Y = Y * 5

X_train, X_test, Y_train, Y_test = train_test_split(
    *arrays: X, Y, test_size=0.2, random_state=13, shuffle=True)

model = Sequential()
model.add(Masking(mask_value=0, input_shape=(max_frames, 162)))
model.add(Conv1D(filters=32, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(Conv1D(filters=32, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=64, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(Conv1D(filters=64, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=128, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(Conv1D(filters=128, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
# model.add(Conv1D(filters=256, kernel_size=5, strides=1,
#                  padding="same", activation="relu"))
# model.add(Conv1D(filters=256, kernel_size=5, strides=1,
#                  padding="same", activation="relu"))
# model.add(BatchNormalization())
# model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(rate=0.2))
# Flatten the results to feed into a DNN
```



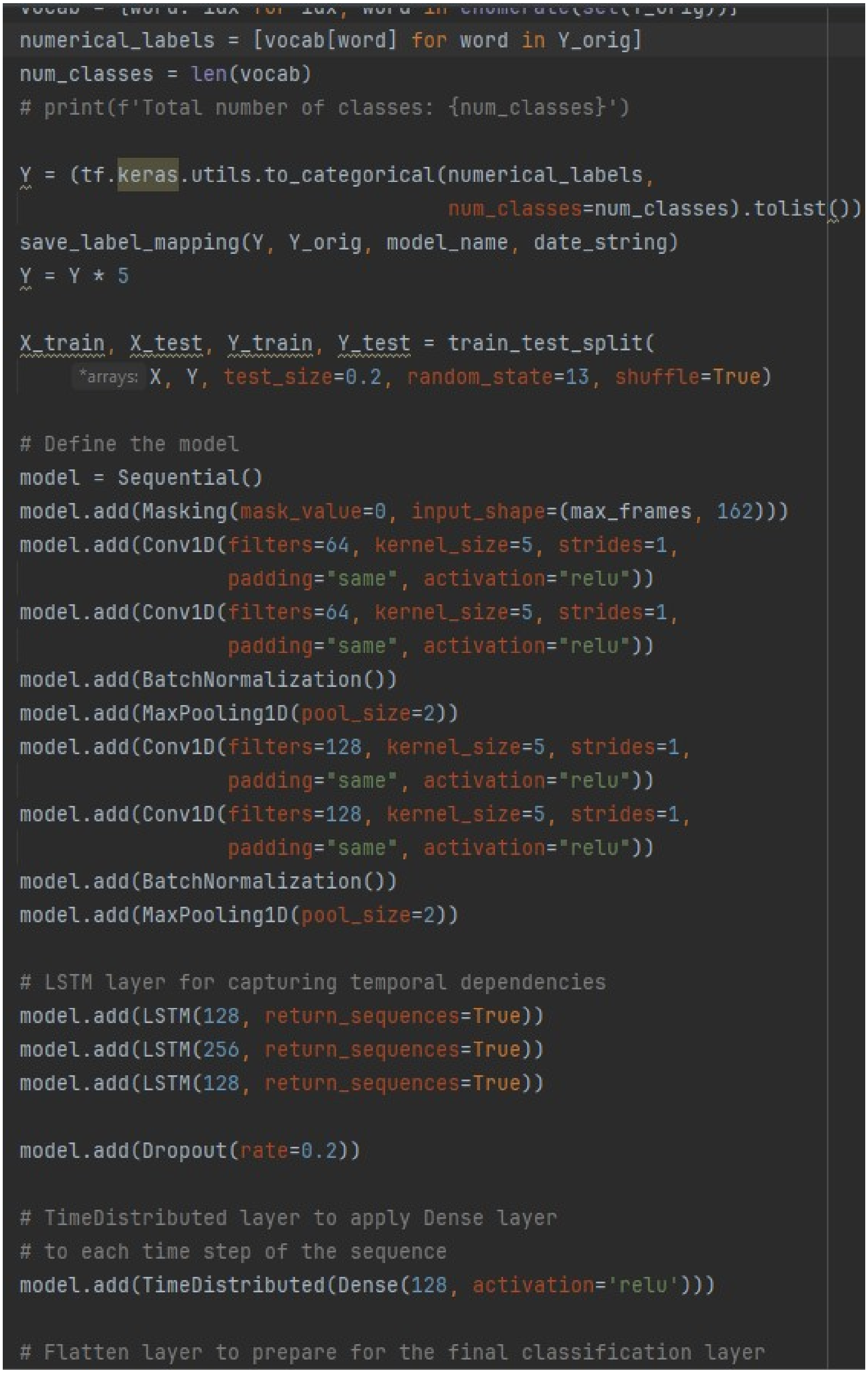
```
# Flatten the results to feed into a DNN
model.add(Flatten())
# model.add(GlobalAveragePooling1D())
# 512 neuron hidden layer
model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
model.summary()

early_stopping = EarlyStopping(
    monitor='val_loss', patience=50, restore_best_weights=True)

start_time = time.time()
model.fit(X_train, Y_train,
          epochs=200, batch_size=34,
          validation_data=(X_test, Y_test),
          # shuffle=True,
          callbacks=[early_stopping]
          )
total_time = (time.time() - start_time)/60
print(total_time)

filename = f'../models/models_{date_string}/{model_name}.keras'
os.makedirs(os.path.dirname(filename), exist_ok=True)
model.save(filename)
```

\

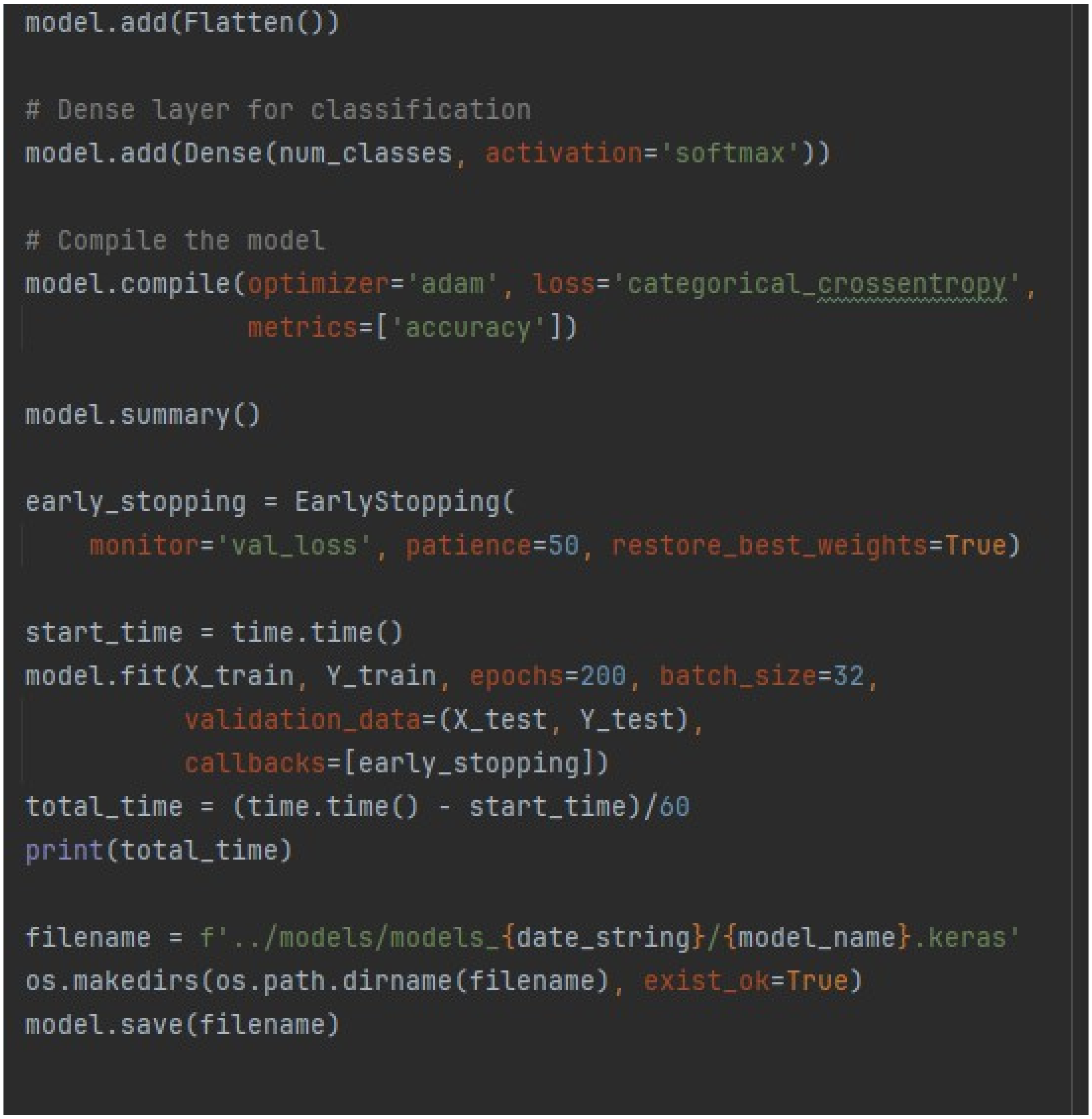
Code for training the CNN-LSTM hybrid model.



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv1D, MaxPooling1D, LSTM,
                                     TimeDistributed, Dense,
                                     BatchNormalization, Masking,
                                     Dropout, Flatten,
                                     GlobalAveragePooling1D, Reshape)
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from DTW_align import dtw_speed_normalization as dtw_transform
from backend.data_utils import (get_features, get_labels,
                                spatial_jittering, scale_sequence,
                                flip_horizontal, rotate_sequence,
                                pad_video_sequences,
                                save_label_mapping)
import tensorflow as tf
import time
import os


def train_cnn_lstm_model(corpus_size, DTW, date_string):
    model_name = f"CNN_LSTM_{'DTW_' if DTW else ''}{corpus_size}"
    X = get_features(f'../features_lmks/features_{corpus_size}.txt')
    Y_orig = get_labels(f'../features_lmks/labels_{corpus_size}.txt')

    if DTW:
        X = dtw_transform(X, Y_orig)

    # data augmentation
    X_spat_jittered = spatial_jittering(X)
    X_scaled = scale_sequence(X)
    X_flipped = flip_horizontal(X)
    X_rotated = rotate_sequence(X)
    X = X + X_spat_jittered + X_scaled + X_flipped + X_rotated

    max_frames = max(len(seq) for seq in X)
    X = pad_video_sequences(X, max_frames)
    # print(f'{max_frames} and {len(X[40])}')

    vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
```

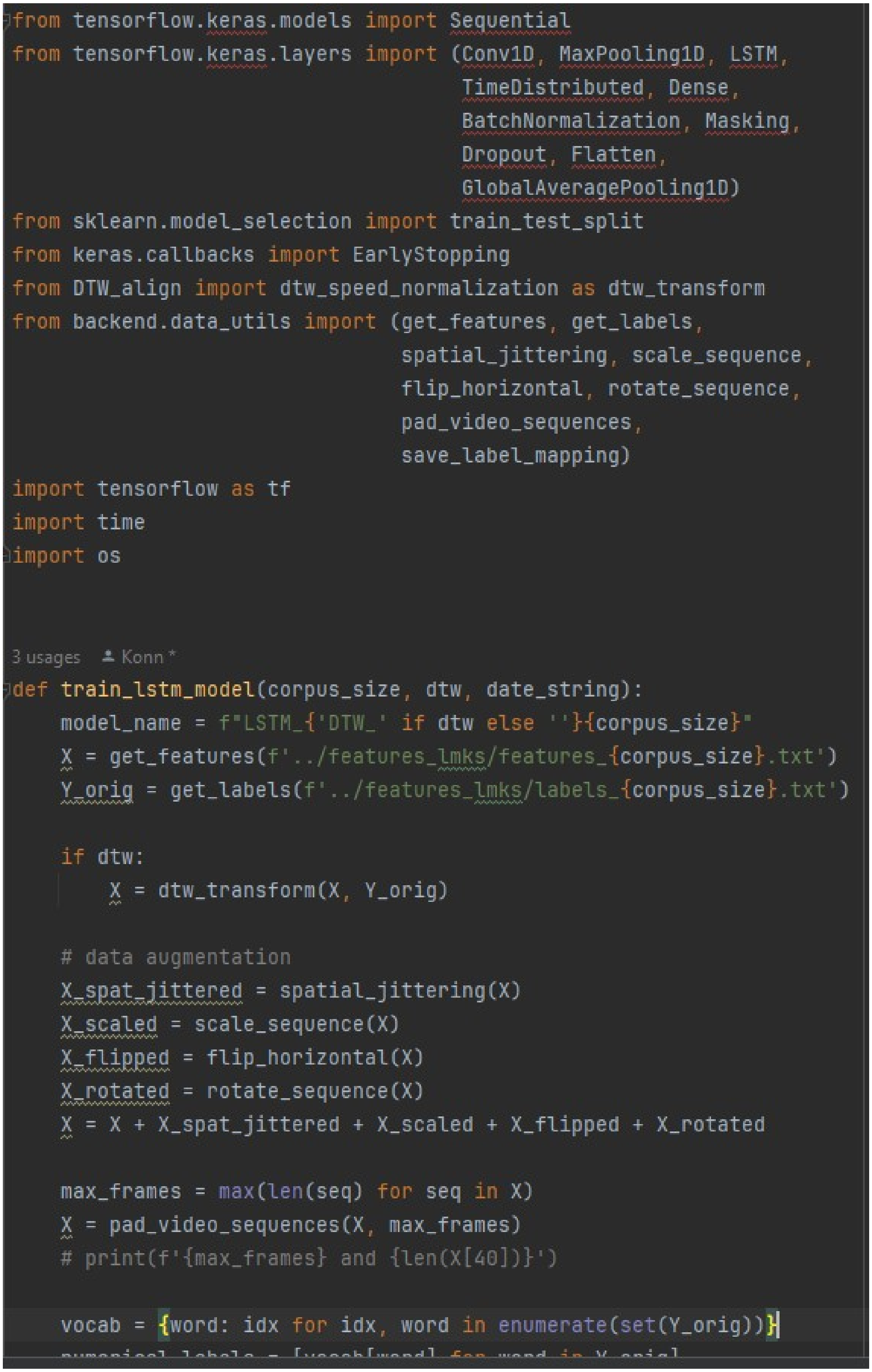


```
vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
numerical_labels = [vocab[word] for word in Y_orig]
num_classes = len(vocab)
# print(f'Total number of classes: {num_classes}')

Y = (tf.keras.utils.to_categorical(numerical_labels,
                                   num_classes=num_classes).tolist())
save_label_mapping(Y, Y_orig, model_name, date_string)
Y = Y * 5

X_train, X_test, Y_train, Y_test = train_test_split(
    *arrays: X, Y, test_size=0.2, random_state=13, shuffle=True)

# Define the model
model = Sequential()
model.add(Masking(mask_value=0, input_shape=(max_frames, 162)))
model.add(Conv1D(filters=64, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(Conv1D(filters=64, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=128, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(Conv1D(filters=128, kernel_size=5, strides=1,
                 padding="same", activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2))

# LSTM layer for capturing temporal dependencies
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(256, return_sequences=True))
model.add(LSTM(128, return_sequences=True))

model.add(Dropout(rate=0.2))

# TimeDistributed layer to apply Dense layer
# to each time step of the sequence
model.add(TimeDistributed(Dense(128, activation='relu')))

# Flatten layer to prepare for the final classification layer
```

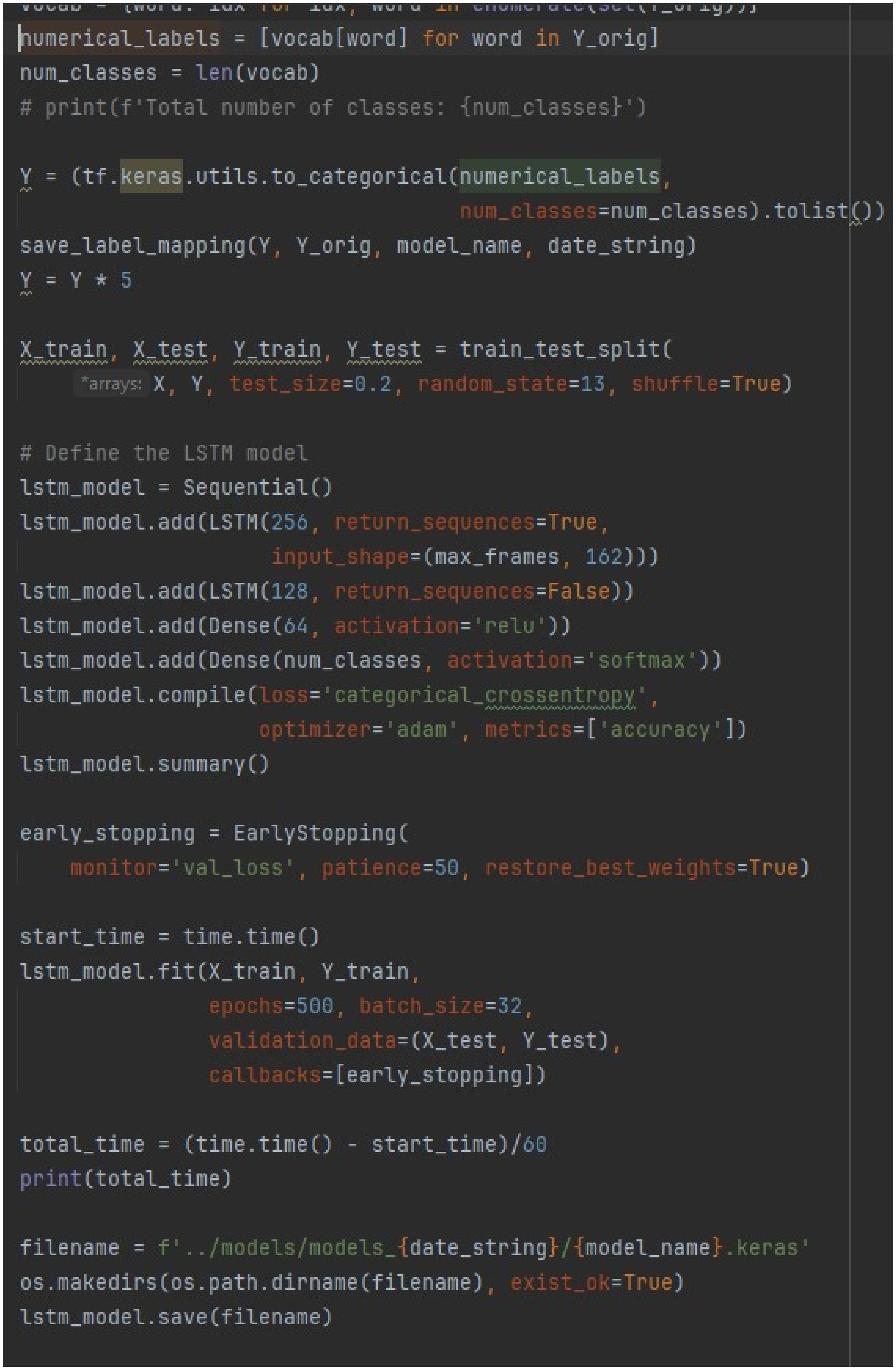


```
model.add(Flatten())

# Dense layer for classification
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

early_stopping = EarlyStopping(
    monitor='val_loss', patience=50, restore_best_weights=True)

start_time = time.time()
model.fit(X_train, Y_train, epochs=200, batch_size=32,
          validation_data=(X_test, Y_test),
          callbacks=[early_stopping])
total_time = (time.time() - start_time)/60
print(total_time)

filename = f'../models/models_{date_string}/{model_name}.keras'
os.makedirs(os.path.dirname(filename), exist_ok=True)
model.save(filename)
```

Code for training the LSTM model.

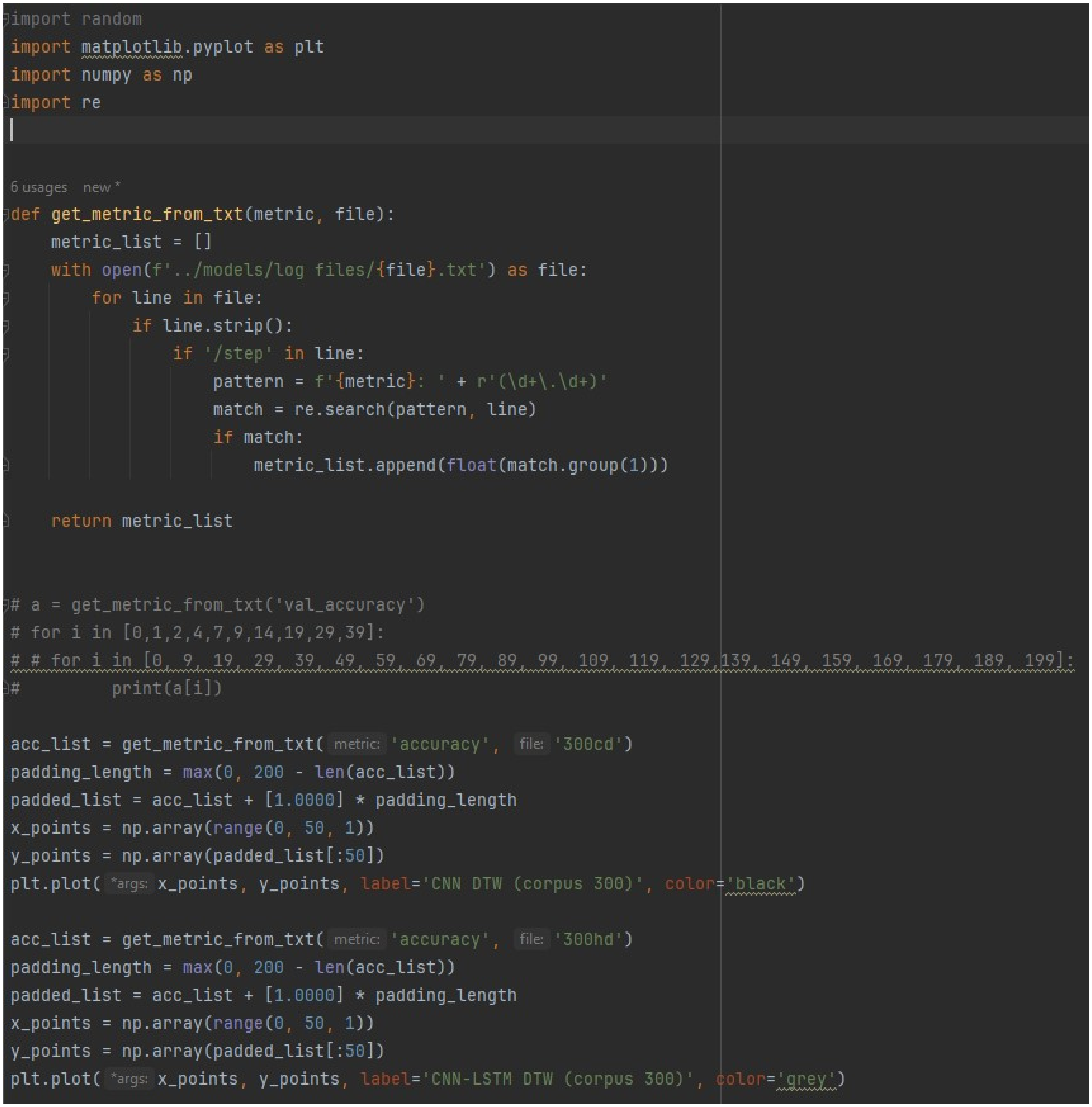


```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv1D, MaxPooling1D, LSTM,
                                     TimeDistributed, Dense,
                                     BatchNormalization, Masking,
                                     Dropout, Flatten,
                                     GlobalAveragePooling1D)
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from DTW_align import dtw_speed_normalization as dtw_transform
from backend.data_utils import (get_features, get_labels,
                                spatial_jittering, scale_sequence,
                                flip_horizontal, rotate_sequence,
                                pad_video_sequences,
                                save_label_mapping)
import tensorflow as tf
import time
import os


3 usages   Konn *
def train_lstm_model(corpus_size, dtw, date_string):
    model_name = f"LSTM_{'DTW_' if dtw else ''}{corpus_size}"
    X = get_features(f'../features_lmks/features_{corpus_size}.txt')
    Y_orig = get_labels(f'../features_lmks/labels_{corpus_size}.txt')

    if dtw:
        X = dtw_transform(X, Y_orig)

    # data augmentation
    X_spat_jittered = spatial_jittering(X)
    X_scaled = scale_sequence(X)
    X_flipped = flip_horizontal(X)
    X_rotated = rotate_sequence(X)
    X = X + X_spat_jittered + X_scaled + X_flipped + X_rotated

    max_frames = max(len(seq) for seq in X)
    X = pad_video_sequences(X, max_frames)
    # print(f'{max_frames} and {len(X[40])}')

    vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
    numerical_labels = [vocab[word] for word in Y_orig]
```

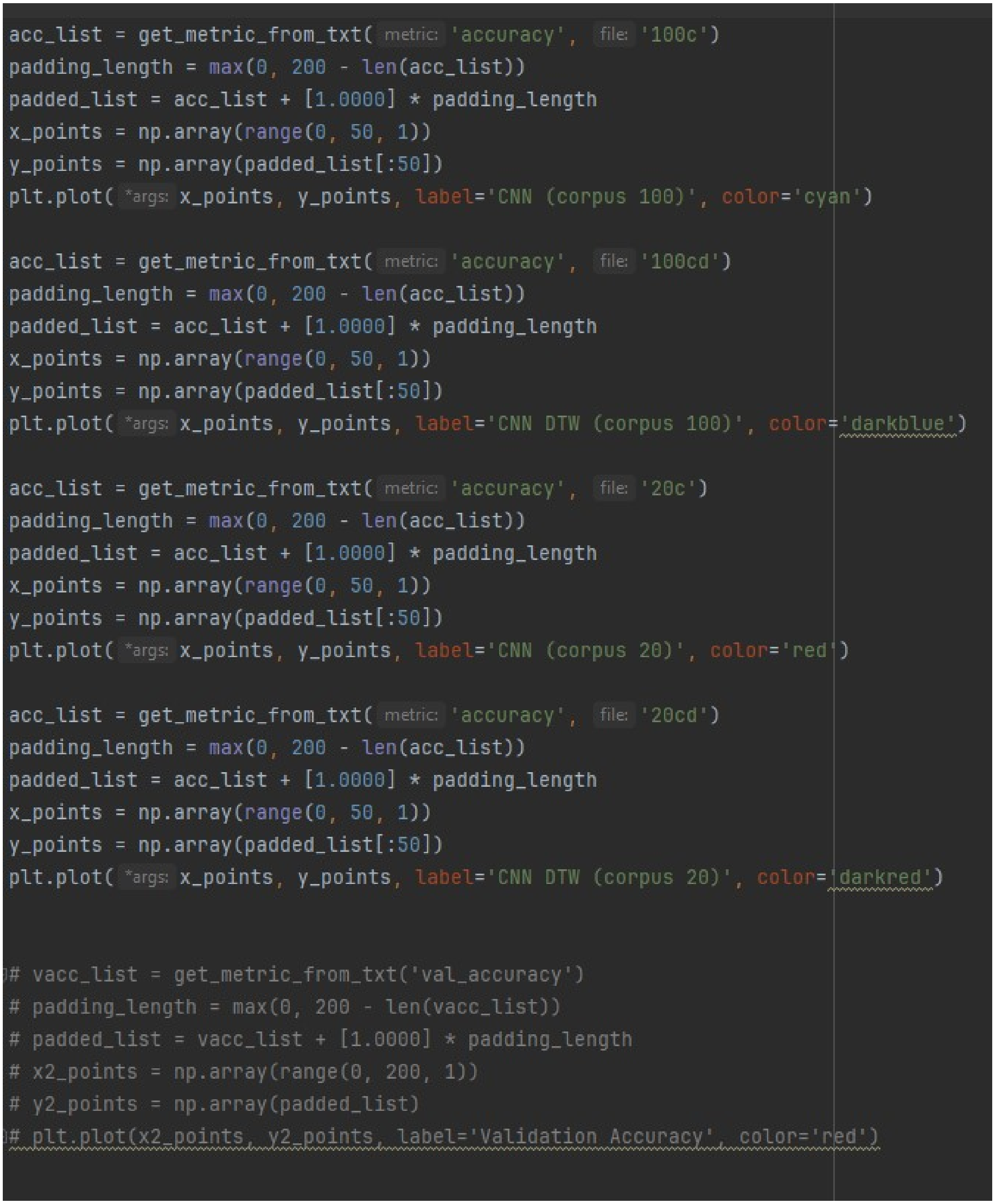


```
vocab = {word: idx for idx, word in enumerate(set(Y_orig))}
numerical_labels = [vocab[word] for word in Y_orig]
num_classes = len(vocab)
# print(f'Total number of classes: {num_classes}')

Y = (tf.keras.utils.to_categorical(numerical_labels,
                                   num_classes=num_classes).tolist())
save_label_mapping(Y, Y_orig, model_name, date_string)
Y = Y * 5

X_train, X_test, Y_train, Y_test = train_test_split(
    *arrays: X, Y, test_size=0.2, random_state=13, shuffle=True)

# Define the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(256, return_sequences=True,
                    input_shape=(max_frames, 162)))
lstm_model.add(LSTM(128, return_sequences=False))
lstm_model.add(Dense(64, activation='relu'))
lstm_model.add(Dense(num_classes, activation='softmax'))
lstm_model.compile(loss='categorical_crossentropy',
                   optimizer='adam', metrics=['accuracy'])
lstm_model.summary()

early_stopping = EarlyStopping(
    monitor='val_loss', patience=50, restore_best_weights=True)

start_time = time.time()
lstm_model.fit(X_train, Y_train,
               epochs=500, batch_size=32,
               validation_data=(X_test, Y_test),
               callbacks=[early_stopping])

total_time = (time.time() - start_time)/60
print(total_time)

filename = f'../models/models_{date_string}/{model_name}.keras'
os.makedirs(os.path.dirname(filename), exist_ok=True)
lstm_model.save(filename)
```

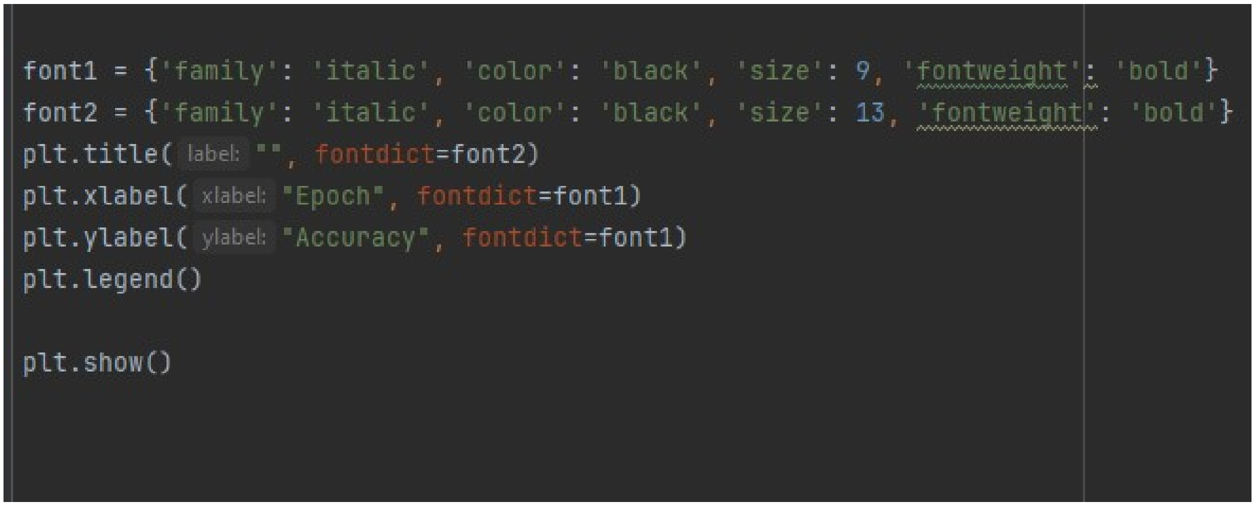
Code for drawing the diagrams for the models' accuracy through epochs.



```python
import random
import matplotlib.pyplot as plt
import numpy as np
import re



def get_metric_from_txt(metric, file):
    metric_list = []
    with open(f'../models/log files/{file}.txt') as file:
        for line in file:
            if line.strip():
                if '/step' in line:
                    pattern = f'{metric}: ' + r'(\d+\.\d+)'
                    match = re.search(pattern, line)
                    if match:
                        metric_list.append(float(match.group(1)))

    return metric_list


# a = get_metric_from_txt('val_accuracy')
# for i in [0,1,2,4,7,9,14,19,29,39]:
# # for i in [0, 9, 19, 29, 39, 49, 59, 69, 79, 89, 99, 109, 119, 129,139, 149, 159, 169, 179, 189, 199]:
#         print(a[i])

acc_list = get_metric_from_txt('accuracy', '300cd')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot(x_points, y_points, label='CNN DTW (corpus 300)', color='black')

acc_list = get_metric_from_txt('accuracy', '300hd')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot(x_points, y_points, label='CNN-LSTM DTW (corpus 300)', color='grey')
```

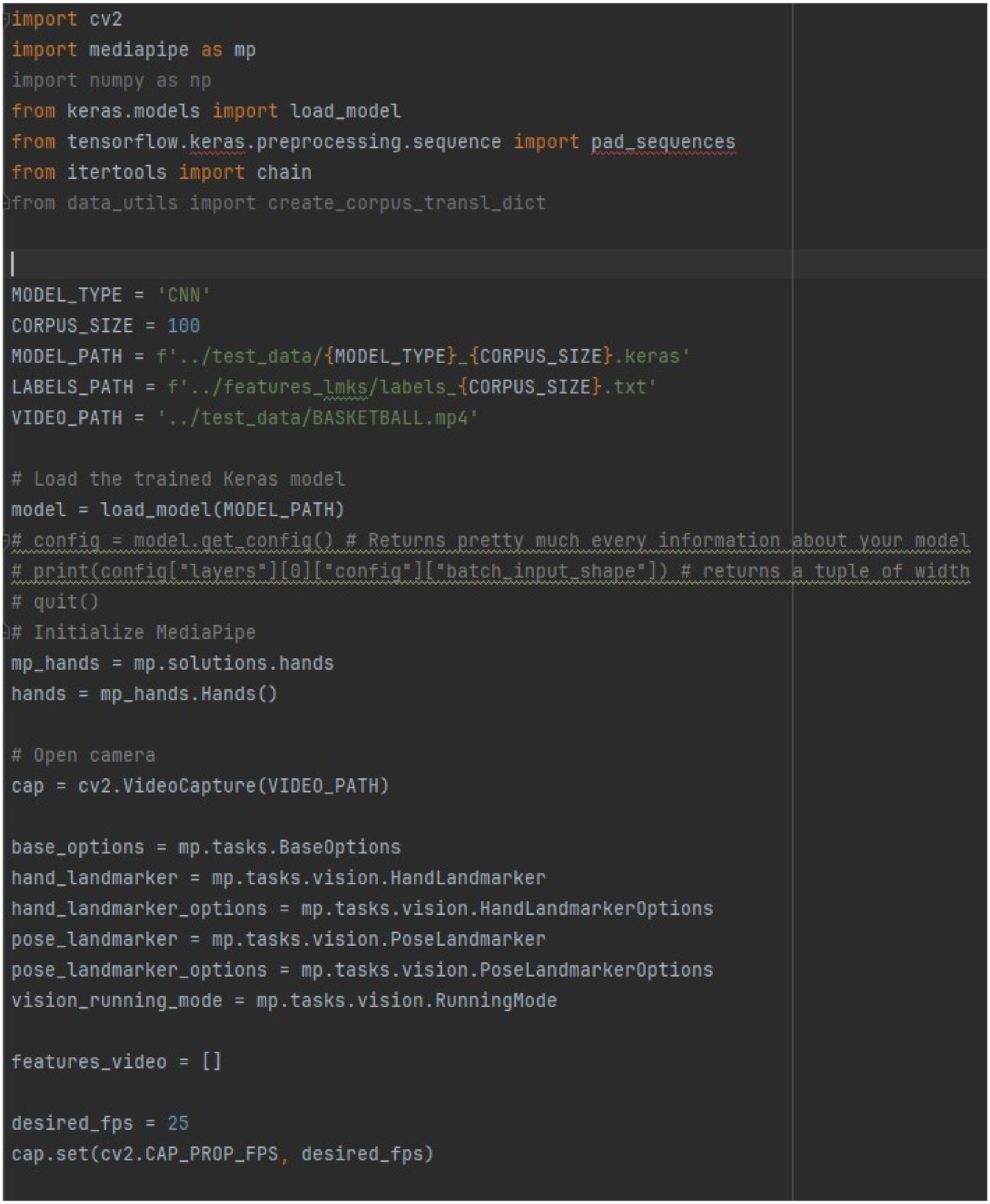


```
acc_list = get_metric_from_txt( metric: 'accuracy',  file: '100c')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot( *args: x_points, y_points, label='CNN (corpus 100)', color='cyan')

acc_list = get_metric_from_txt( metric: 'accuracy',  file: '100cd')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot( *args: x_points, y_points, label='CNN DTW (corpus 100)', color='darkblue')

acc_list = get_metric_from_txt( metric: 'accuracy',  file: '20c')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot( *args: x_points, y_points, label='CNN (corpus 20)', color='red')

acc_list = get_metric_from_txt( metric: 'accuracy',  file: '20cd')
padding_length = max(0, 200 - len(acc_list))
padded_list = acc_list + [1.0000] * padding_length
x_points = np.array(range(0, 50, 1))
y_points = np.array(padded_list[:50])
plt.plot( *args: x_points, y_points, label='CNN DTW (corpus 20)', color='darkred')


# vacc_list = get_metric_from_txt('val_accuracy')
# padding_length = max(0, 200 - len(vacc_list))
# padded_list = vacc_list + [1.0000] * padding_length
# x2_points = np.array(range(0, 200, 1))
# y2_points = np.array(padded_list)
# plt.plot(x2_points, y2_points, label='Validation Accuracy', color='red')
```

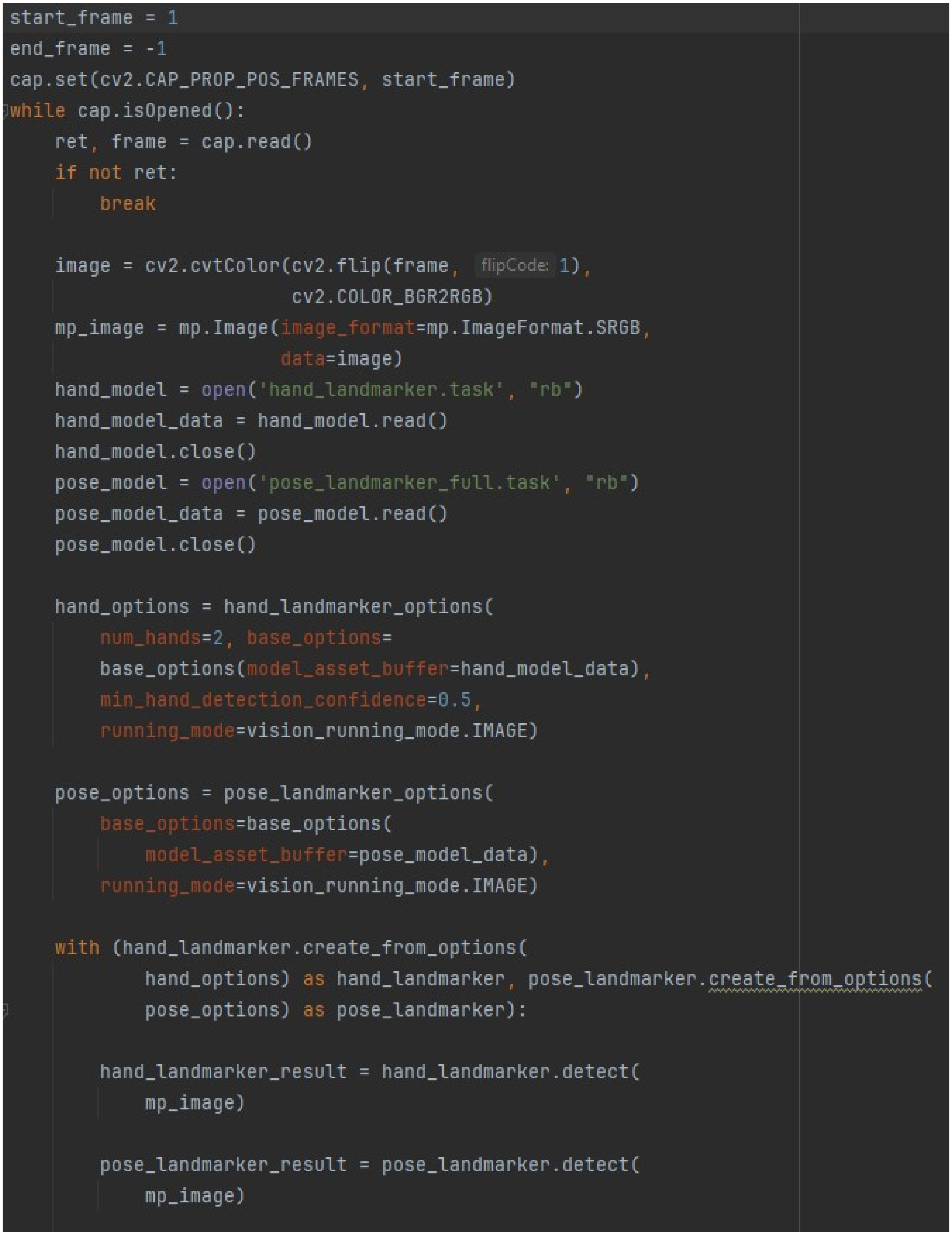


```
font1 = {'family': 'italic', 'color': 'black', 'size': 9, 'fontweight': 'bold'}
font2 = {'family': 'italic', 'color': 'black', 'size': 13, 'fontweight': 'bold'}
plt.title( label: "", fontdict=font2)
plt.xlabel( xlabel: "Epoch", fontdict=font1)
plt.ylabel( ylabel: "Accuracy", fontdict=font1)
plt.legend()

plt.show()
```

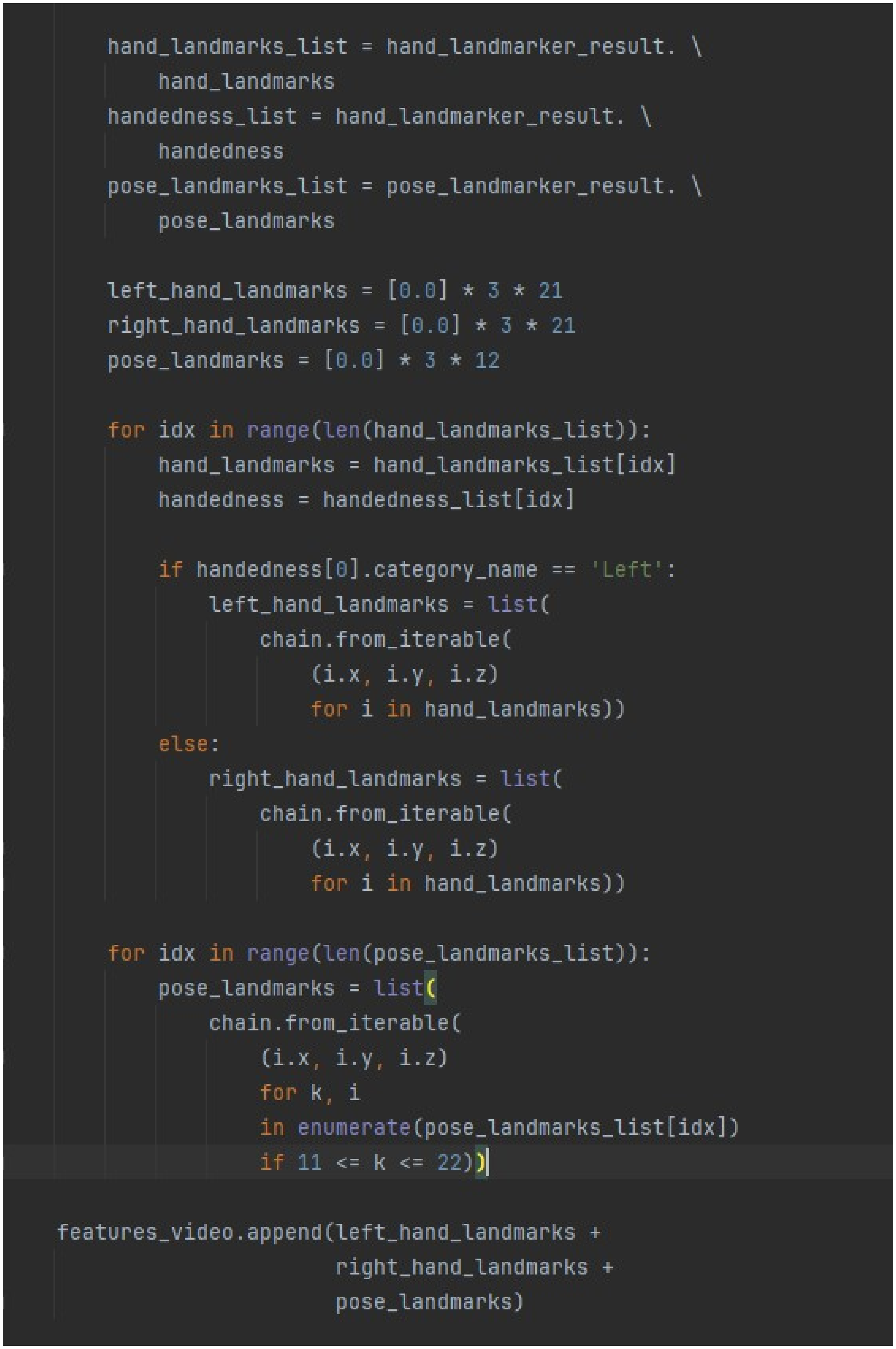
Code for predicting the gesture from a video using a trained model.



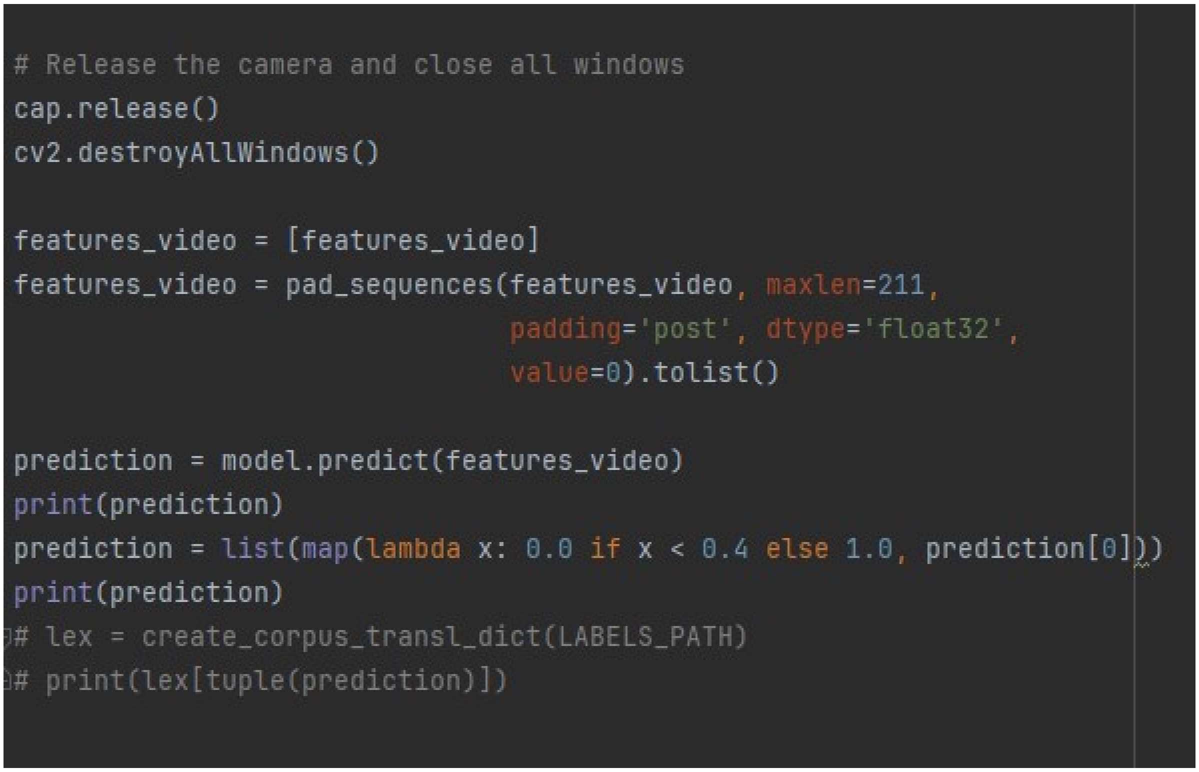
```
import cv2
import mediapipe as mp
import numpy as np
from keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from itertools import chain
from data_utils import create_corpus_transl_dict


MODEL_TYPE = 'CNN'
CORPUS_SIZE = 100
MODEL_PATH = f'../test_data/{MODEL_TYPE}_{CORPUS_SIZE}.keras'
LABELS_PATH = f'../features_lmks/labels_{CORPUS_SIZE}.txt'
VIDEO_PATH = '../test_data/BASKETBALL.mp4'

# Load the trained Keras model
model = load_model(MODEL_PATH)
# config = model.get_config() # Returns pretty much every information about your model
# print(config["layers"][0]["config"]["batch_input_shape"]) # returns a tuple of width
# quit()
# Initialize MediaPipe
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()

# Open camera
cap = cv2.VideoCapture(VIDEO_PATH)

base_options = mp.tasks.BaseOptions
hand_landmarker = mp.tasks.vision.HandLandmarker
hand_landmarker_options = mp.tasks.vision.HandLandmarkerOptions
pose_landmarker = mp.tasks.vision.PoseLandmarker
pose_landmarker_options = mp.tasks.vision.PoseLandmarkerOptions
vision_running_mode = mp.tasks.vision.RunningMode

features_video = []

desired_fps = 25
cap.set(cv2.CAP_PROP_FPS, desired_fps)
```



```
start_frame = 1
end_frame = -1
cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    image = cv2.cvtColor(cv2.flip(frame, flipCode: 1),
                         cv2.COLOR_BGR2RGB)
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB,
                        data=image)
    hand_model = open('hand_landmarker.task', "rb")
    hand_model_data = hand_model.read()
    hand_model.close()
    pose_model = open('pose_landmarker_full.task', "rb")
    pose_model_data = pose_model.read()
    pose_model.close()

    hand_options = hand_landmarker_options(
        num_hands=2, base_options=
        base_options(model_asset_buffer=hand_model_data),
        min_hand_detection_confidence=0.5,
        running_mode=vision_running_mode.IMAGE)

    pose_options = pose_landmarker_options(
        base_options=base_options(
            model_asset_buffer=pose_model_data),
        running_mode=vision_running_mode.IMAGE)

    with (hand_landmarker.create_from_options(
            hand_options) as hand_landmarker, pose_landmarker.create_from_options(
            pose_options) as pose_landmarker):

        hand_landmarker_result = hand_landmarker.detect(
            mp_image)

        pose_landmarker_result = pose_landmarker.detect(
            mp_image)
```

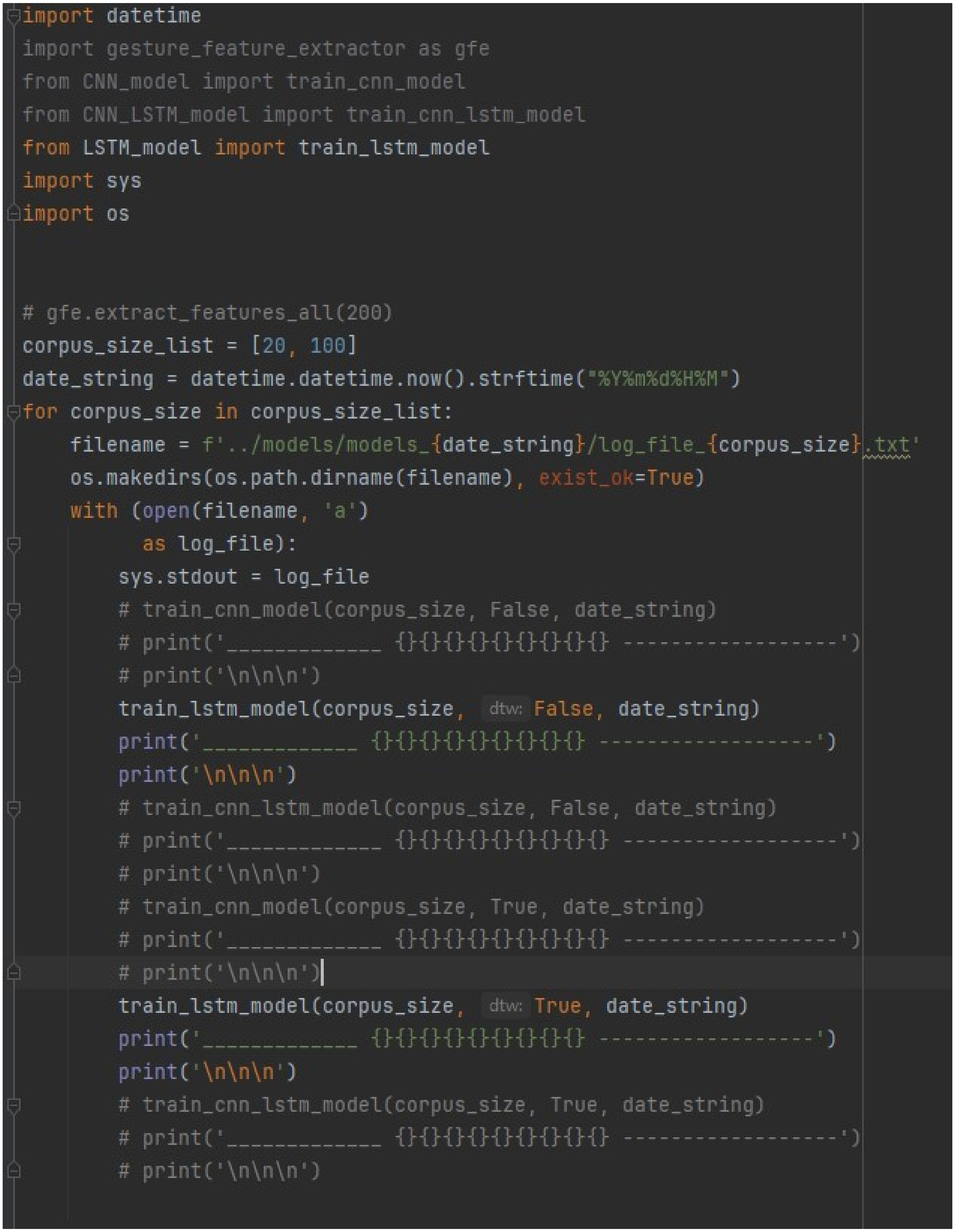


```
        hand_landmarks_list = hand_landmarker_result. \
            hand_landmarks
        handedness_list = hand_landmarker_result. \
            handedness
        pose_landmarks_list = pose_landmarker_result. \
            pose_landmarks

        left_hand_landmarks = [0.0] * 3 * 21
        right_hand_landmarks = [0.0] * 3 * 21
        pose_landmarks = [0.0] * 3 * 12

        for idx in range(len(hand_landmarks_list)):
            hand_landmarks = hand_landmarks_list[idx]
            handedness = handedness_list[idx]

            if handedness[0].category_name == 'Left':
                left_hand_landmarks = list(
                    chain.from_iterable(
                        (i.x, i.y, i.z)
                        for i in hand_landmarks))
            else:
                right_hand_landmarks = list(
                    chain.from_iterable(
                        (i.x, i.y, i.z)
                        for i in hand_landmarks))

        for idx in range(len(pose_landmarks_list)):
            pose_landmarks = list(
                chain.from_iterable(
                    (i.x, i.y, i.z)
                    for k, i
                    in enumerate(pose_landmarks_list[idx])
                    if 11 <= k <= 22))

    features_video.append(left_hand_landmarks +
                          right_hand_landmarks +
                          pose_landmarks)
```



```
# Release the camera and close all windows
cap.release()
cv2.destroyAllWindows()

features_video = [features_video]
features_video = pad_sequences(features_video, maxlen=211,
                               padding='post', dtype='float32',
                               value=0).tolist()

prediction = model.predict(features_video)
print(prediction)
prediction = list(map(lambda x: 0.0 if x < 0.4 else 1.0, prediction[0]))
print(prediction)
# lex = create_corpus_transl_dict(LABELS_PATH)
# print(lex[tuple(prediction)])
```

Code for extracting the features and training the proposed models for a given corpus size.



```
import datetime
import gesture_feature_extractor as gfe
from CNN_model import train_cnn_model
from CNN_LSTM_model import train_cnn_lstm_model
from LSTM_model import train_lstm_model
import sys
import os


# gfe.extract_features_all(200)
corpus_size_list = [20, 100]
date_string = datetime.datetime.now().strftime("%Y%m%d%H%M")
for corpus_size in corpus_size_list:
    filename = f'../models/models_{date_string}/log_file_{corpus_size}.txt'
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with (open(filename, 'a')
            as log_file):
        sys.stdout = log_file
        # train_cnn_model(corpus_size, False, date_string)
        # print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        # print('\n\n\n')
        train_lstm_model(corpus_size, dtw: False, date_string)
        print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        print('\n\n\n')
        # train_cnn_lstm_model(corpus_size, False, date_string)
        # print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        # print('\n\n\n')
        # train_cnn_model(corpus_size, True, date_string)
        # print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        # print('\n\n\n')
        train_lstm_model(corpus_size, dtw: True, date_string)
        print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        print('\n\n\n')
        # train_cnn_lstm_model(corpus_size, True, date_string)
        # print('_____________ {}{}{}{}{}{}{}{}{} ------------------')
        # print('\n\n\n')
```