Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

A mobile application based on machine learning and music therapy principles for post-stroke upper-limb motor recovery

Mireia de Gracia Forés

Supervisor: Rafael Ramírez

July 2023

Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

A mobile application based on machine learning and music therapy principles for post-stroke upper-limb motor recovery

Mireia de Gracia Forés

Supervisor: Rafael Ramírez

July 2023

Table of Contents

Dedication

A la meva estimada família, tant la de naixement com la que he escollit amb el cor,

Per la vostra presència constant i el suport incessant que m'heu ofert amb generositat. Per estimar-me incondicionalment, en els moments de llum i en els de foscor, i per exercir una paciència que sembla no tenir límits.

Gràcies per encoratjar-me sempre, fins i tot quan el camí davant meu es torna difús.

Amb profunda gratitud.

Abstract

A stroke is a medical condition caused by a disruption in blood flow to the brain. This can lead to difficulties with everyday activities and movement. Music therapy is a promising new alternative to traditional rehabilitation methods. This therapy uses sound's natural properties to enhance stroke recovery, improve motor skills and stimulate neural plasticity. This approach motivates people on both a physical and emotional level. Software tools developed to date to aid in motor recovery after stroke rely mainly on external mechanisms and specific hardware components. This limitation restricts the potential scope of these tools. This study aims to examine the effectiveness and mechanisms of using a mobile application with machine learning algorithms and music therapy principles as a complementary intervention for post-stroke motor recovery. This research project has resulted in the development of a mobile app, based on the widely used Fugl Meyer Assessment. The application uses Vision Framework from Apple and a custom Activity Classification CoreML machine learning model to detect an individual's position in a seated posture. It has also been integrated with XCode. The application generates an audio cue when a user successfully completes one of the Fugl-Meyer Assessment activities. To train the model, 340 clips of a variety of exercises have been created. The research sheds light on how this technology can be used to transform neurorehabilitation while also helping to develop accessible and convenient tools that promote stroke motor recovery.

Keywords: music therapy; stroke; machine learning; CoreML; Fugl-Meyer

1. Introduction

A stroke is a condition that happens when the blood flow to the brain gets disrupted. It can cause serious issues that have a substantial impact on how someone lives their life and how they feel. Traditional rehabilitation methods have been widely used to support motor recovery. However, recent advancements have unveiled a promising approach known as music therapy. This innovative therapeutic technique harnesses the powerful influence of music on the brain, stimulating the brain's ability to adapt (neural plasticity), facilitating the reacquisition of motor skills, and improving overall stroke motor recovery. Music therapy offers a comprehensive rehabilitation approach that actively engages and motivates individuals on both physical and emotional levels. This thesis aims to investigate the underlying mechanisms and effectiveness of utilizing a mobile app as a supplementary intervention in music therapy for stroke motor recovery. By shedding light on the potentially transformative nature of this technology within the field of neurorehabilitation practices, the study endeavours to contribute to the development of accessible and convenient tools for promoting stroke motor recovery.

1.1. Motivation

The motivation behind this thesis lies in the recognition of the pressing need for innovative and accessible approaches to stroke motor recovery. While music therapy has shown promising results, traditional methods often require specialized resources, specific hardware and trained professionals, limiting their widespread implementation. This study intends to solve these issues and provide stroke survivors with a practical, affordable, and easily accessible solution by suggesting a mobile app as an alternative to traditional music therapy. Utilizing mobile technology for neurorehabilitation holds significant potential to reach a bigger population and offer individualized treatment because of the rising prevalence of smartphones and their incorporation into daily life. This thesis seeks to advance neurorehabilitation techniques by examining the potential and efficacy of a mobile app as an additional intervention in stroke motor recovery, ensuring that survivors of stroke have access to cutting-edge and effective tools for their recovery process.

1.2. Objectives

The objectives of this thesis are outlined as follows:

- To develop an easy-to-use mobile app that can be utilized by stroke survivors, allowing independent practice of motor exercises and therapeutic activities.
- To explore in which ways music therapy fundamentals can be applied to this new technology and include it.
- To investigate the feasibility and effectiveness of a mobile app as a supplement in stroke motor recovery, which facilitates motor improvement without the need for additional hardware or special equipment.
- To include a machine learning model in the app for seated poses and exercises detection.
- To gather feedback and insights regarding the acceptability, usability, and perceived benefits of the mobile app in comparison to conventional music therapy methods.

By achieving these objectives, this thesis aims to provide valuable evidence and insights into the practicality of a mobile app as an alternative approach for stroke motor recovery. Additionally, it hopes to enhance teletherapy possibilities, which would improve stroke survivors' access to neurorehabilitation treatments in the long run.

2. State of the art

2.1 Stroke

Strokes are one of the most serious forms of cerebrovascular disease that occur when blood flow to a part of the brain is interrupted or reduced, depriving brain cells of oxygen and nutrients. As a result, brain cells begin to degenerate in a matter of minutes, which causes a high mortality rate, a high rate of morbidity, and a considerable degree of disability, all of which have a significant influence on the lives of the survivors. Patients may develop aphasia, dysphagia, cognitive impairment, motor dysfunction, mood issues, and other consequences as a result of the stroke. Damage to the parts of the brain that control language results in language disorders such as aphasia. Aphasia can make it challenging for people to find words, form sentences, understand spoken or written language, read, and write. The medical term dysphagia refers to the difficulty swallowing liquids and food which can lead to dehydration, malnutrition and aspiration pneumonia. Cognitive recovery includes memory and learning ability, sensory and perceptual disorders, and attention. According to the degree of affectation on the patient, motor dysfunction can range from numbness or weakness of the limbs to paralysis or spasms.
This thesis will focus primarily on motor dysfunction. The damage to the nervous system in many cases is not completely recoverable and cannot be completely reversed. That’s why early intervention and rehabilitation is fundamental to restoring impaired neurological function, which depends on the brain's plasticity and functional reorganization (Huang et al., 2021).

2.1.1 Stroke motor recovery evaluation

There are 5 stages to stroke recovery. The first two weeks are defined as the acute stage of stroke. From 3 to 11 weeks, sub-acute stage. From 12 to 24 weeks is the early chronic stage and, finally, more than 24 weeks is the chronic stage of the stroke.
The evaluation possibilities for stroke recovery are wide as it depends on what is to be assessed. For the evaluation of motor recovery, which will be the focus of this thesis, an examination of the patient's physical abilities and functional status is needed. Some common tests used for this purpose are Fugl-Meyer Assessment (FMA), Motor Assessment Scale (MAS), Action Research Arm Test (ARAT), Barthel Index… All these tests provide valuable information about the patient's motor recovery state after a stroke and can help with the creation and development of an effective rehabilitation program. In this thesis Fugl-Meyer Assessment (FMA) will be considered the standard as it is currently what is being used at Hospital Forum Mar by the music therapy professionals.

2.2 Music Therapy

The practice of using music to treat conditions like depression, pain, anxiety, and cognitive function is known as music therapy. The fundamental idea behind this therapy is that because music involves so many different neuronal connections and functions, it has a perceptible effect on the brain. For instance, the left part of the brain is responsible for lyric comprehension and rhythm distinction, while the right part is focused on melody analysis. During a session, a trained music therapist assesses the individual's needs and develops a personalized treatment plan. This plan may include passive activities, such as listening to the music of the user's liking, making them feel comfortable and safe. Or alternatively, active activities, such as singing, playing with instruments, and moving along with the music. Studies have shown that music therapy is a potential strategy for stroke patients' rehabilitation since it can help to lessen the stroke's numerous side effects. It is a reliable, affordable, practical, and simple-to-apply solution. This, together with its high patient acceptance, explains why this therapy is becoming more and more popular

among staff members in hospitals, rehabilitation centers, and nursing homes. (Braun Janzen et al., 2022).

2.2.1 Music Therapy for stroke recovery

Music therapy has various applications for stroke recovery due to music's inherent properties and its connectivity with the brain. Early music intervention may promote long-term neuroplasticity changes in sensory and perceptual processes that facilitate cognitive function recovery. For example, vocal music can be helpful to improve memory recovery whereas mindful music might assist with relaxation and concentration. In the same way that vocal and singing exercises are advised for speech recovery in dysphagia, rebuilding cortical and subcortical structural networks in the dominant hemisphere of language is the main objective in aphasia treatment. The brain regions that are engaged while speech and singing activities overlap, according to neuroimaging research. Melodic Intonation Therapy (MIT), which involves singing words and phrases while engaging in daily tasks and using a musical tune, was born as a result of this. Patients who are unable to talk may be able to sing words.

Regarding the thesis's primary focus on motor function recovery, a range of techniques can be implemented, among which is Rhythmic Auditory Stimulation (RAS) for path cadence. An auditory rhythmic cue is utilized here in the form of repetitive isochronous pulses (e.g., a metronome) or metrically accentuated music matching the individual's cadence. As the cues are gradually increased or decreased by 5-10%, they provide anticipatory time cues that allow movements to be planned and prepared, helping therefore to regulate timing and pace. Additionally, Music-Based Interventions (MBI) have been shown to improve upper extremity motor function recovery. These iMBI techniques include Music-Supported Therapy (MST), a standard rehabilitation technique that employs a keyboard or electronic drum to enhance both movement kinematics and motor function in the subacute stage of a stroke. This is an enjoyable activity for patients and helps with movement coordination, auditory-motor coupling and integration. Music therapists also use Patterned Sensory Enhancement (PSE) to improve timing, duration, direction, and force of movements by playing musical instruments. With the creation of

musical patterns for different movements, actions that would not be rhythmical by nature, for instance getting dressed, are structured and regulated.

Instead, in Therapeutic Instrumental Music Performance (TIMP), which will be of great importance for this thesis, musical instruments are simply feedback, either tactile, visual or auditory. Individuals' bodies are strategically arranged with instruments at strategic locations. With this, various movements can be trained.

The impact of music therapy on stroke rehabilitation has been gradually substantiated in clinical practice (Xu et al., 2022).

2.3 Technology-enhanced stroke rehabilitation

The goal of this chapter is to give a general review of the cutting-edge technology utilized in stroke therapy while highlighting both its advantages and disadvantages.

Robotic-assisted therapy has garnered significant attention in stroke rehabilitation. Robotic devices offer high-intensity, repetitive movements that can facilitate motor recovery. These devices provide precise control, allowing therapists to target specific muscle groups and customize interventions according to individual needs. Examples of robotic-assisted therapy devices include robotic exoskeletons, end-effector devices, and robotic-assisted gait training systems. Studies have shown promising results in improving upper and lower limb function, reducing muscle tone, and enhancing overall motor recovery (Hesse et al., 2003; Mehrholz et al., 2015).

Virtual Reality (VR) and Augmented Reality (AR) provide experiences that facilitate motor learning and neuroplasticity while performing rehabilitation exercises in virtual environments, increasing users' engagement and adherence to therapy. VR generates computer-generated 3D simulations, while AR overlays virtual elements in the real world. Additionally, this method can provide real-time feedback, assessment tools, and data analytics to track progress and customize interventions (Laver et al., 2017; Saposnik et al., 2016).

Brain-Computer Interfaces (BCIs) establish a direct communication pathway between the brain and external devices, bypassing impaired motor pathways. With this, stroke survivors control robotic devices or computer programs using their brain activity, promoting motor recovery and enhancing functional independence (Ang et al., 2014;

Cervera et al., 2018). Electroencephalography (EEG) and functional near-infrared spectroscopy (fNIRS) are commonly used to detect and monitor brain signals.

Wearable sensors like accelerometers and gyroscopes provide home-based therapy and continuous monitoring. These tools offer precise movement measurements, allowing therapists to evaluate motor function from a distance. For example, machine learning algorithms can be used to analyse the acquired data and provide individualized feedback and improve treatment plans. Through remote monitoring and telerehabilitation, this technology removes geographic restrictions and improves access to care. (Porciuncula et al., 2018).

Games have also been used to transform traditional rehabilitation exercises into interactive and enjoyable activities. This motivates stroke survivors and enhances their engagement in therapy. By incorporating elements like rewards, competition, and progression, games promote repetitive practice and motor learning. They can be combined with other technologies such as VR and motion sensors to create immersive and challenging rehabilitation experiences (Lohse et al., 2014; Saposnik et al., 2016).

Lastly, Telerehabilitation and Mobile Health (mHealth) solutions enable remote rehabilitation services, improving access to care and supporting home-based rehabilitation. Telecommunication technologies, video conferencing, and mobile apps allow therapists to provide guidance, monitor progress, and offer remote interventions. Stroke survivors can perform exercises at home while receiving real-time feedback and support from healthcare professionals. These technologies have the potential to reduce healthcare costs, increase patient autonomy, and improve long-term outcomes (Dodakian et al., 2017; Tchero et al., 2018) . Several software apps for stroke recovery are accessible on the internet, catering to both patients and professionals. While some programs provide informative resources for stroke patients and their families, like Codigo Ictus and Essential Anatomy 3, others are developed to support recovery treatment, such as Constant Therapy for speech therapy and CPA for image-based communication.

The primary focus of this thesis is to examine applications that target the motor recovery of the upper limb, for instance, Dexteria and CloudRehab. Dexteria (Sawant et al., 2020) is an app with several game-like exercises that require the user to use their hand to improve pencil grip, work with both hands and follow lines. It does not include arm

exercises, focusing only on hand and finger strength. CloudRehab (Rodriguez-Prunotto & Cano-de-la-Cuerda, 2018) facilitates patients to review their recorded rehabilitation session, while simultaneously observing themselves doing the same exercise on the other half of the screen. The recorded session is then transmitted to the designated doctor or therapist for analysis, who subsequently provides feedback to the patient. But again, there is no feedback from the app to the patient, just a mirroring service.

When it comes to music therapy apps for stroke recovery, most are for relaxation, meditation, and music selection. There are other options available, all of which require external hardware to be used, such as a piano, sensors (GotRhythm, Jintronix, MedRhythms) or specifically built an instrument for the application such as the one developed in (Segura et al., 2021)
The key point here is that, currently, there is no app for motor recovery after a stroke that makes use of music therapy techniques and requires no external hardware or specific materials.

2.3.1 Machine Learning Models in Mobile Applications

The incorporation of Machine Learning and Artificial Intelligence in mobile applications has given cell phones the ability to convert text to speech, interpret gestures, and identify objects and people in images. All of these new possibilities have piqued the interest of developers in many domains, including healthcare.

Machine Learning methodologies can be grouped into four main categories. Supervised Learning entails training the algorithm with labelled inputs and outputs, followed by predicting the output with the test dataset. Some examples of Supervised Learning include Decision Trees, Random Forests, K-Nearest Neighbours, Linear Regression, Logistic Regression, Neural Networks, and Deep Learning. On the other hand, Unsupervised Learning involves the algorithm learning similarities, patterns, or differences to classify unlabelled data. Examples of this are k-Means clustering and Mean-shift. There is also something in between, Semi-supervised learning, where labelled and unlabelled data are combined throughout the training phase to produce either a category or numerical output

(regression). Another different approach can be Reinforcement Learning, in which the algorithm learns from its environment. This methodology learns from its experiences in the environment and once it has encountered the entire range of possible states, it can make decisions based on its acquired knowledge.

In order to incorporate Machine Learning in mobile applications, there are two main architectures. Server-side architecture, where everything is done on to the server through a web service in which the information is sent to be analysed. The deployment of the model on the server endows all clients with instantaneous updates on the model, which may have occurred through reinforcement learning during the evaluation on the server. This approach is ideal for applications that demand a precise model while utilizing the device's minimum amount of space and CPU power. Contrary, on client-side architecture, a copy of the trained model exists on the device and it is executed locally to calculate the result, enabling its use even in the absence of internet connectivity. This approach is particularly advantageous for image processing, as the evaluation is more direct due to the absence of network latency (Ganesan, 2022).

For the purposes of the present thesis, a client-side architecture will be employed. This approach guarantees that the patients' images will not be transmitted to any online server, which is a crucial factor in secure medical data treatment. It is crucial to acknowledge that the models accessible for implementation in mobile applications are subject to constraints in terms of space and processing power. Hence, even though there might be several models available online for the purpose of this thesis, specific models designed explicitly for a mobile environment must be selected.
From all the frameworks available such as TensorFlow, Keras, Scikit and ML Kit. Core ML (with the use of the Vision framework from Apple) and XCode offer the infrastructure to develop and implement an app with a machine learning model for iOS platforms.

2.4. MoveNet

MoveNet is a state-of-the-art model evolved by Google AI, designed for real-time human pose estimation. Pose estimation refers to the procedure of detecting the position and orientation of 17 key body parts (key points) in an image or video. This model architecture prioritizes velocity and performance without compromising accuracy, contrary to traditional pose estimation models that frequently require powerful computing hardware. MoveNet comes in two variants: Movenet Lightning, designed for ultra-fast inference and suitable for applications that require actual-time feedback, and Movenet Thunder, a bigger version designed for programs where higher accuracy is preferred and some computational overhead is acceptable.

The output of the model in both cases is a tensor with 17 xy coordinates of the body points and the confidence scores for the key point.

3. Methods and material

3.1. Materials

Name	**Comment**
iPhone 12	iOS 16.5.1, 68GB
Macbook Pro 2020	MacOS Ventura 13.3.1, Apple M1, 16GB RAM

Due to the availability of this material and the impossibility to use app simulators for a live camera application, the platform for which the application has been coded is iOS.

3.1.1. Fugl-Meyer Assessment (FMA)

In the Fugl-Meyer Assessment (FMA), motor recovery after stroke can be quantitatively measured. Its scale, Fugl-Meyer scale, was the first developed quantitative evaluation for sensorimotor recovery measurement in stroke. It consists of 100 points based on different exercises that involve mobility for the upper and lower extremities (Gladstone et al., 2002).

A score from 0 to 2 is assigned to each of these tasks. Patients with a score of 0 cannot perform the exercise, whereas those with a maximum score are able to perform the task completely. The scores of all sections are then added together to obtain a total score.

In this thesis, the Upper Extremity exercises (66 points if done perfectly, 33 tasks) proposed by FMA are used as an inspiration for the development of the exercises in the mobile application. Mentioned exercises are listed below for the reader's comprehension.

Upper Extremity (66 points)	Lower Extremity (34 points)
Shoulder retraction	Hip flexion
Shoulder elevation	Hip extension (supine)
Shoulder abduction	Hip adduction (supine)
Shoulder abduction to 90 degrees	Knee flexion (supine)
Shoulder adduction/internal rotation	Knee flexion (sitting)
Shoulder external rotation	Knee flexion (standing)
Shoulder flexion 0–90 degrees	Knee extension (supine)
Shoulder flexion 90–180 degrees	Ankle dorsiflexion (supine)
Elbow flexion	Ankle dorsiflexion (sitting)
Elbow extension	Ankle dorsiflexion (standing)
Forearm supination	Ankle plantar flexion (supine)
Forearm pronation	Heel-shin speed
Forearm supination/pronation (elbow at 0 degrees)	Heel-shin tremor
Forearm supination/pronation (elbow at 90 degrees, shoulder at 0 degrees)	Heel-shin dysmetria
Hand to lumbar spine	Knee reflex
Wrist flexion/extension (elbow at 0 degrees)	Hamstring reflex
Wrist flexion/extension (elbow at 90 degrees)	Ankle reflex
Wrist extension against resistance (elbow at 0 degrees)	
Wrist extension against resistance (elbow at 90 degrees)	
Wrist circumduction	
Finger flexion	
Finger extension	
Extension of MCP joints, flexion of PIPs/DIPs	
Thumb adduction	
Thumb opposition	
Grasp cylinder	
Grasp tennis ball	
Finger-nose speed	
Finger-nose tremor	
Finger-nose dysmetria	
Finger flexion reflex	
Biceps reflex	
Triceps reflex	

Figure 1: Exercises for Fugl-Meyer evaluation

3.1.2. XCode

XCode is an Apple-developed integrated development environment (IDE) that provides a visual interface where developers can design the user interface, write code, and debug applications. It includes a simulator to test and preview the app's behavior with different iOS hardware (iPhone, iPad, iPod, Mac, all different generations). Any CoreML model can also be integrated into the project. Once integrated, the model can be utilized to make real-time predictions within the iOS app. Among the programming languages that this software offers, the app has been coded in Swift.

3.1.3. Create ML – Activity Classifier

Create ML is the MacOS software that has all Apple CoreML algorithms available. The Action Classifier Model in Create ML is a valuable tool for developing accurate machine

learning models that can recognize activities effectively. This model employs Apple Vision framework to detect 19 body landmarks, which are essential points or features within an activity that help differentiate and categorize actions. After detecting these landmarks, a neural network is used for classification of the different activities.

The output of the model in terms of data includes all possible categories or movements that have been detected (labels), as well as the confidence or probability of each of them, selecting the one with highest confidence as the predicted action.

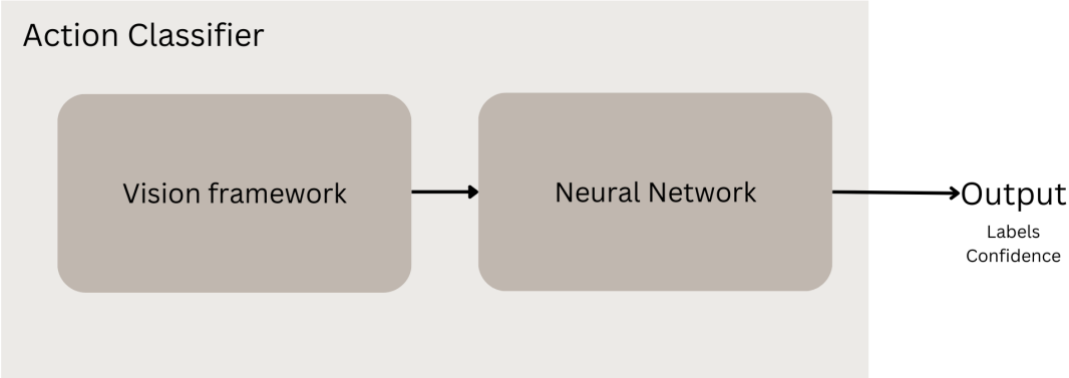


Figure 2: Simplified diagram of Action Classifier from CoreML

For each of these landmarks, both x and y coordinates are computed. For example, in Fugl-Meyer activities, landmarks can include the hand, arm, or shoulder positions during movement. By tracking these landmarks, the model learns to associate specific patterns with different activities.

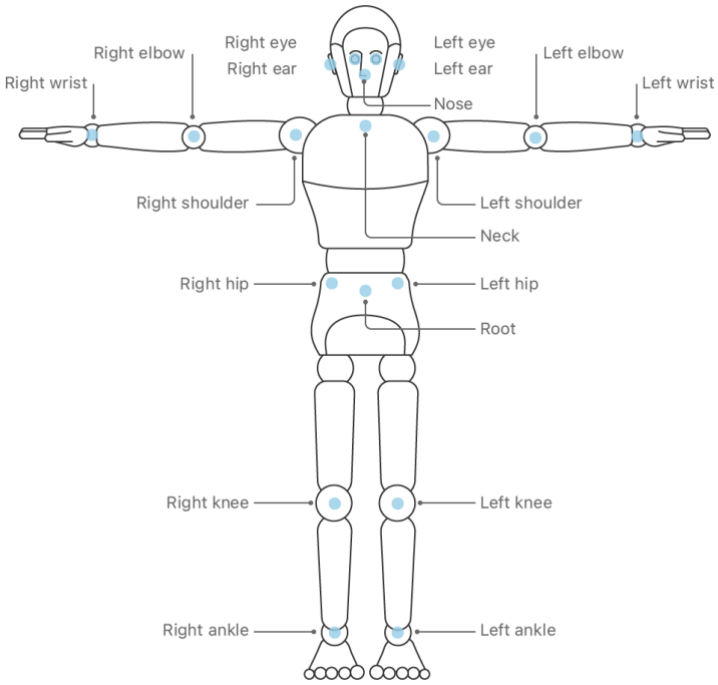


Figure 3: Vision framework body landmarks detection.

3.2. Selection of the posture detection framework

It is worth noting that existing machine learning models for posture recognition are primarily designed for a full-body perspective, specifically focusing on individuals in a standing position. That can be due to the dataset chosen for the pretraining of these models. However, stroke survivors and individuals undergoing Fugl Meyer assessment activities are typically seated. The existing models for seated positions aim at cameras located laterally to the participants, thus are not fitting for the context of the app in this thesis.

One of the main tasks of this thesis has been to decide which framework to use to obtain the landmarks due to this limitation.

As MoveNet is one of the state-of-the-art models for body landmarks detection, the first approach was to use this framework for this purpose. The idea was to first obtain these points and then use different classifiers for activity classification. The code for obtaining these captures is available in Appendix – MoveNet screen captures code.

Figure 4: Body landmarks detected with MoveNet Thunder

Figure 5: Body landmarks detected with MoveNet Lightning

Given these results, MoveNet cannot be used for the purpose of this application, as body landmarks are not well detected, probably because of the seated position. A classifier is highly dependent on the input data. In this case, the input data would not be representative of the different actions.

Otherwise, landmark detection in CoreML is more accurate for the required task because of the use of Vision, thus the final classification would be more precise. Therefore, the justification for the use of this framework for the development of the app.

Figure 6: Body landmarks detected with CoreML using Vision

3.3. Data collection: Creating the Dataset

The development of an activity detection model for Fugl Meyer activities needs of the creation of a video dataset showcasing the specific target activities. As there is no dataset available online for such a purpose, this section delineates the primary steps involved in constructing the dataset.

As the scope of this thesis is to provide a proof of concept for an app utilizing music therapy in motor stroke recovery, it is important to note that not all exercises encompassed by the Fugl Meyer assessment will be available in the dataset. Rather, the focus will be on two specific exercises: the movement of the hand from the knee to the ear, as well as shoulder lateral flexion spanning the range of 0 to 90 degrees. Both are done with both hands.

Figure 7: Ear exercise example

Figure 8: 90lateral exercise example

Regarding the video recording setup, the camera's positioning should allow for a complete view of the participant's body in the frontal camera frame. Video samples should be recorded at a frame rate of 30 frames per second (fps), as this aligns with the configuration of the developed app's camera input. If recorded at any other frame rate, this info would need to be updated both in the model configuration and in the corresponding parts of the code regarding data input.

Videos have been recorded using the frontal camera of an iPhone 12 set to 30fps, then cropped with Adobe Premiere to 1 second duration clips, manually labelled and organized in a directory structure categorizing the activities performed. Independently of how many exercises are to be detected, an extra “negative” category with examples of no-movements (i.e. resting position) needs to be added for the proper function of CreateML classifier.

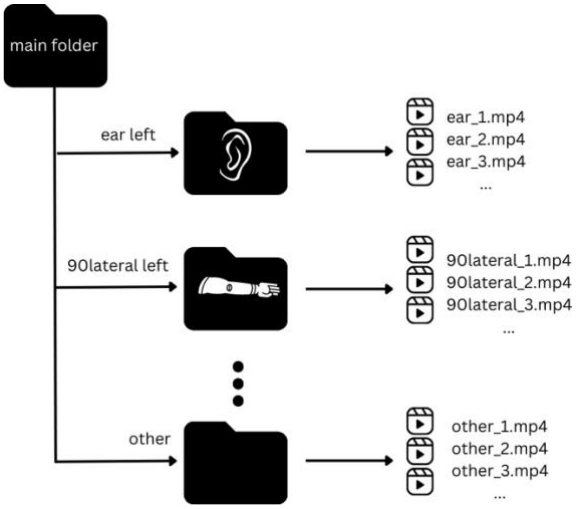


Figure 9: Database folder distribution for training and testing

The resulting dataset consists of 340 clips:

Table 1: Dataset clips and classes distribution

Activity label	**Total clips**
Ear left	68
Ear right	69
90lateral left	67
90lateral right	68
Other (resting)	68
Total	340

3.4. Training CoreML

The Action Classifier Model undergoes a training process where it learns from a labelled dataset, where each example is associated with a specific activity category. Once trained, it can be used to predict the activity category of new, unseen examples. By feeding the input data, such as an image or video frame, into the model's neural network, the model analyses the input and generates a prediction indicating the most likely activity category.

CreateML automatically divides data for training and validation. Optionally, a set of clips can also be selected for testing evaluation. The number of iterations, the prediction window size and the framerate of the input data can be selected. Data augmentation is

also possible thanks to the horizontal flip feature that it offers, training then the same action for both sides.

The accuracy of the trained model can be measured using metrics like precision, recall, and F1 score. This helps determine the effectiveness of the model in recognizing and classifying activities. A preview of the model behaviour is also available in the same program for quick testing, either with pre-recorded videos or live camera input.

The dataset has been automatically split for training and validation. For testing evaluation, around 20% of the dataset has been separated.

3.5. FuglMeyerApp – the result app

The resulting app from this thesis, FuglMeyerApp, is an iOS app coded in Swift using XCode 14.3.

What the app essentially does is described in the following diagram. First, a live video feed from the frontal camera of the user is sent to the app. This input feed is then analysed within a defined prediction window. For every frame in this window, with the use of Vision framework, body landmarks are detected. With said landmarks, then the trained model can predict which activity the user is doing. If the confidence for such prediction is higher than a threshold, the prediction is considered valid, and a sound is played as feedback. The sound varies according to the activity that is being completed. As a help for the user to understand with the app functioning, a preview layer is added as a unique output in the user's screen, in which the user can see on top of themselves in real-time the body landmarks that have been detected.

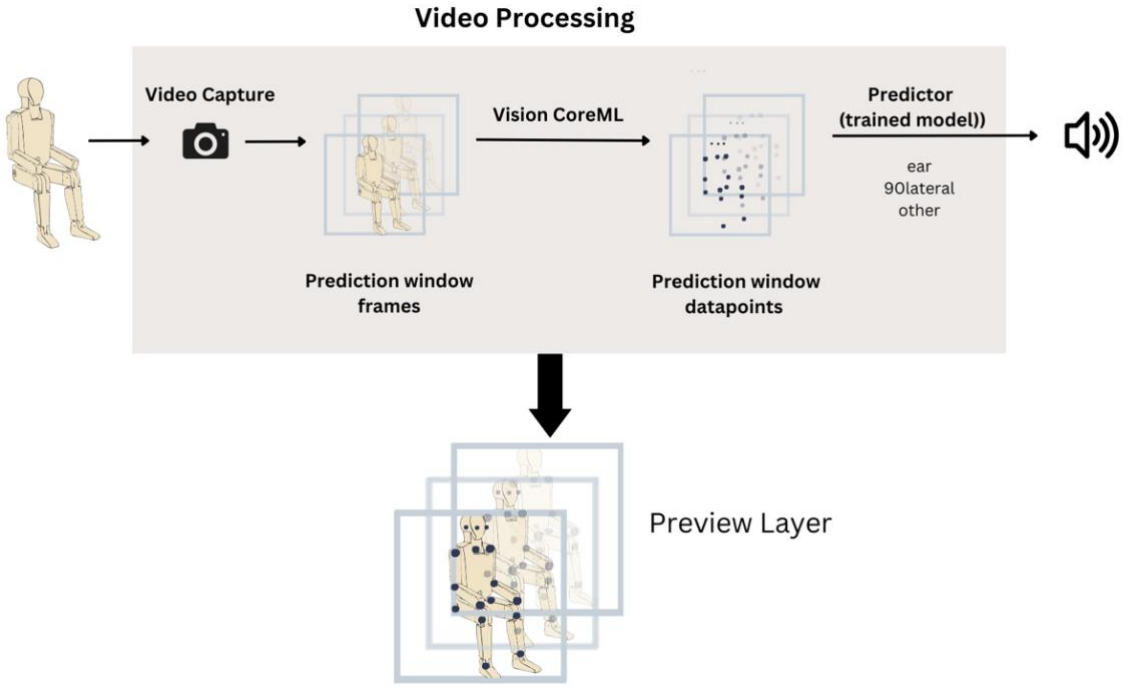


Figure 10: Proposed application scheme

The dependencies used for this purpose are Vision, AVFoundation and CoreML.
The code for the main parts of the app can be found in Appendix – Main APP Code Parts. Also in Appendix, a README file is included. The following documentation serves as a comprehensive guide to understanding the app's structure and behaviour.

3.5.1.Structure

The modular structure and organized codebase ensure scalability and maintainability, allowing for further enhancements and customization. The app is now functioning for five classes (ear right, ear left, 90 lateral right, 90 lateral left and others). But more activities could be easily added to the app with no issue due to its structure. The main parts of the app:

- View Components: These components handle how the app looks and how users interact with it. They include the main view controller, storyboard files, and various UI elements like buttons, labels, and video preview layers.
- Capture and Processing Components: These components are responsible for capturing video data from the device's camera and processing it to estimate body

positions and predict actions. The VideoCapture class handles video capturing, while the Predictor class processes the data and makes predictions.

- Machine Learning Model: At the heart of the action prediction functionality lies a trained machine learning model. In this app, the FMClassifier (Fugl-MeyerClassifier) model is used, which analyzes body pose observations to label detected actions.
- Audio Feedback Components: These components provide real-time auditory cues to the user based on the detected actions. To accomplish this, we use AVAudioPlayer instances to play sound files that deliver ear and lateral cues.

3.5.2. Capturing data

The VideoCapture class is responsible for capturing video data from the device's frontal camera and passing it to the predictor for further processing. It utilizes the AVFoundation framework to set up an AVCaptureSession, which manages the input (camera) and output (video data). The main behavior of this class includes:

- Initialization: Upon initialization, the VideoCapture class sets up the capture session and adds the AVCaptureVideoDataOutput, which is responsible for receiving video frames.
- Capture session Start: The startCaptureSession() method is called to start the capture session and activate the video capture.
- Sample Buffer Delegate: The class conforms to the AVCaptureVideoDataOutputSampleBufferDelegate protocol, allowing it to receive sample buffers containing video frames. The output capture method is triggered whenever a new sample buffer is available. In this method, the predictor instance is invoked to estimate body poses based on the received sample buffer.

3.5.3. Predictor

The Predictor class is responsible for analyzing the body pose observations received from the video capture and making predictions about the detected actions. It utilizes a trained machine learning model (**FMClassifier**) to label the actions and calculates the

confidence level for each prediction. It also delegates the recognized points to the ViewController for visualization. The key behaviors of this class include:

- Body Pose Estimation: The estimation method is called when a new sample buffer is received. It uses Vision and a human body pose request to process the sample buffer and obtain human body observation instances representing body poses.
- Recognized Points Processing: For each received human body observation, the recognized body points are extracted and transformed to display coordinates. The transformed points are then sent to the delegate through the didFindNewRecognizedPoints() method for visualization purposes.
- Action Labeling: The stored human body observation instances are used as input to the FMClassifier model to predict the actions. The labeled action and its confidence level are sent to the delegate through the didLabelAction() method.
- Observation Storage: The class maintains a window of human body observation instances to ensure a continuous stream of data for action prediction. When a new observation is received, it is stored in the window. If the window exceeds the defined prediction window size, the oldest observation is removed.

The recognized points are superposed to the input camera information, showing the points on top of the user's body. This is done so the user can be certain that their extremities are being properly captured by the app and to notice situations such as lag, delays or points out of position which could occur in case of error.

3.5.4. Adding sound

The audio feedback functionality is implemented using AVAudioPlayer instances. Based on the labelled actions and their confidence levels, the app provides real-time auditory cues to the user upon action completion. The specific audio files and playback logic can be customized and extended according to the specific requirements of the rehabilitation program. This means, if more activities were added in the future, more sounds could also be introduced easily in the code.

3.5.5. Accessibility and landing screen

As the target for this application is people who are recovering from a stroke, some considerations are needed regarding the layout of the application. The app itself should be as easy as possible to use. Because of this, no registration is needed, and the idea is to have an app that is "plug and play". As soon as the app is started, after granting access to the camera, the frontal camera is enabled and body key points are displayed on top of the user's view. From that moment, the app can detect the exercises. This is the easiest scenario that could be implemented. Even if users do not have others nearby to help them set up the app, the app should be easy enough to use. Exercises demanded on the app should also be easy to do and the app behavior should be understandable by the user.
A landing screen showing how to setup properly the camera should suffice for using the application.

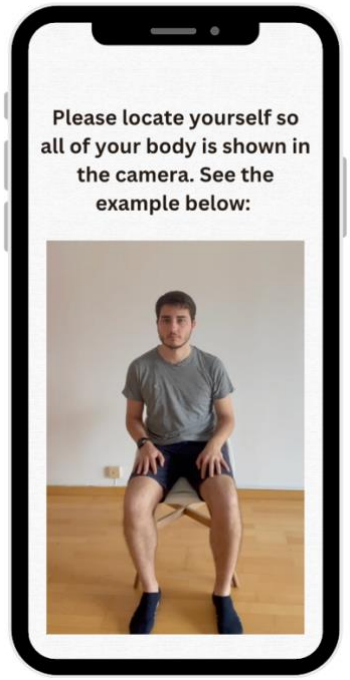


Figure 11: Example of landing screen

3.6. Music Therapy principles in the app

One of the key motivations of building this kind of application was to incorporate music therapy principles into it, with the idea to come up with a solution that does not require external hardware. For this, different music therapy techniques were studied, deciding how it could be included in an app of this nature to help with post-stroke motor recovery. The approach that seemed more convenient was Therapeutic Instrumental Music Performance (TIMP), in which instruments are used as auditory, tactile or visual feedback. TIMP emphasizes the therapeutic benefits of actively engaging with musical instruments, such as promoting coordination and motor control. In this app, users interact with the "virtual instrument" of their body through their motor movements, creating a dynamic and interactive musical experience. The app leverages the inherent rhythmic elements of music to facilitate motor learning, enhance motivation, and provide a sense of enjoyment and accomplishment.

Upon completion of an action, auditory feedback is generated, akin to the production of sound when playing a percussive musical instrument, which typically requires physical movement. The sounds selected for this demo are drums and percussion clips from Freesound, so that the body of the user feels like a drum set themselves. This decision has been made taking into consideration that not every user might be able to differentiate the pitch or different notes of the same instrument, thus it would be easier to have a variety of instruments that are not precisely melodic for this purpose. This selection also enables users to engage in drumming while participating in music therapy sessions at the hospital, where music therapists commonly utilize the piano or guitar. Furthermore, users can play the drums along with their preferred music at home, or alternatively, practice exercises aimed at creating rhythmic patterns.

3.7. Usability Questionnaire

When designing a therapeutic application, usability is a must. Users should find some benefit in using it and doing the exercises. Also, it should not be difficult to understand what is happening. Otherwise, the contribution of this thesis would be only technical and not practical.

The ideal situation would have been to conduct a clinic study with stroke survivors during some months, a control group who would have been using the app several times a week as a supplement to their normal therapy sessions, and another group who just had the music therapy sessions at the hospital with no extra practice. Progress for patients in both groups could have been tracked with Fugl-Meyer Assessment evaluation. But due to time constraints, this escapes the scope of this thesis and is encouraged to do in the future.

As an alternative, testing sessions have been conducted with volunteers from their 20s to people in their 80s, focusing more on people older than 40 years. In each of these individual sessions, individuals have been given contextual information on what is stroke, the inconveniences and difficulties that stroke survivors might deal with, what is music therapy and how it is used for stroke recovery, specially TIMP. After that, they were asked to use the application for a few minutes and to complete the usability questionnaire that can be found in Appendix – Usability questionnaire. Examples of questions include a rating on how intuitive to use the app is, how easy to do the exercises are, and app's overall behaviour, among others.

4. Results

4.1. CoreML

The Activity classifier model used in the application has been trained as follows:

- Number of iterations: 60
- Augmentation (Horizontal flip): no
- Frame rate: 30 fps
- Action duration: 1 second (30-frame prediction window)

Table 2: Testing evaluation FMClassifier iteration 20

Class	Count	Precision	Recall	F1 Score
Ear right (4)	26	100%	81%	**0.89**
Ear left (3)	26	100%	73%	**0.84**
90lateral right (2)	21	100%	71%	**0.83**
90lateral left (1)	26	100%	65%	**0.79**
Other (0)	15	36%	100%	**0.53**

Here are some examples of the model live output using the camera from the Macbook Pro with the labels predictions and confidence levels. Labels are 3 (ear left, meaning the exercise is done with the left hand) and 2 (90lateral right).

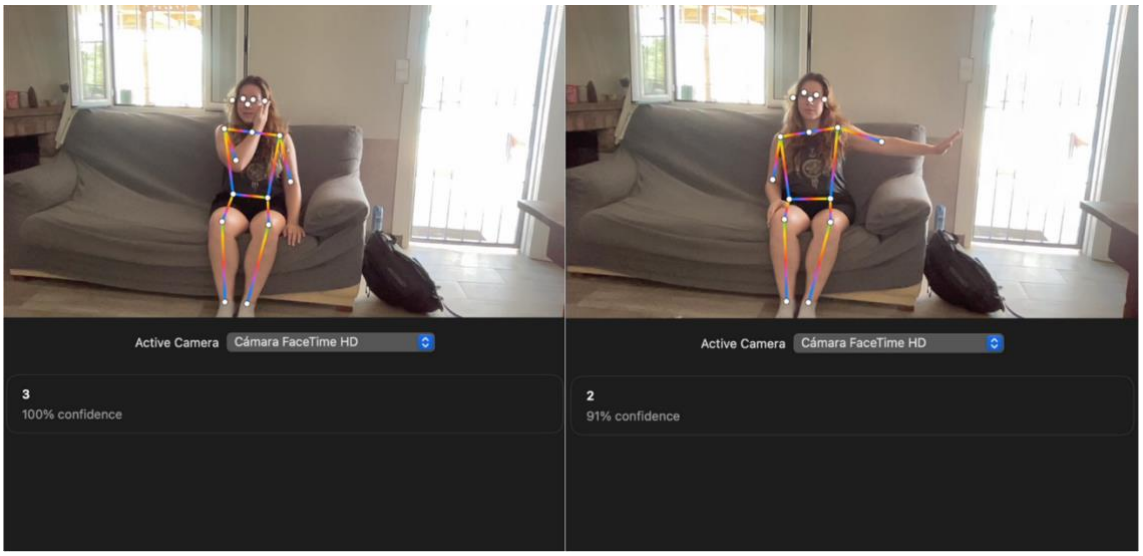


Figure 12: Live Output predictions of the trained CoreML model

4.2. FuglMeyerApp

The outcome of this thesis work culminates in the development of an iOS application that enables the generation of sounds upon successful completion of specific exercises from the Fugl Meyer assessment. Through the utilization of this app, the participant's body assumes the role of a percussive musical instrument. The aim of this app is to help post-stroke survivors with their upper-limb motor recovery.

Figure 13: FuglMeyerApp with custom logo

In order to test the app, an iPhone is connected via cable to a MacbookPro where the XCode project with the application is stored. Then, in XCode with the run option, the iPhone is selected and the app is built into it. All tests regarding the app show real results in terms of timings and processing, because even though the original code is in XCode, the app runs the version that has been built into the iPhone. Posting the app on AppStore would be possible after following some requirements. Still, this would require revision and approval from people at Apple, so this was the faster way to test the application. On the other hand, there is no app hosting fee per se, but an annual fee of 99€ for the Apple Developer Program is needed to publish the app, which is also another constraint.

Figure 14: Screenshot of the iPhone while the app is being used

The app's code as well as other code used in this thesis for results is available in https://github.com/mireiadg/fuglmeyer_recovery.git

4.3. Questionnaire results

The complete results chart can be found in Appendix – Usability Questionnaire Results.

- Demographics:

The age of the volunteers ranged from 24 to 85 years, with a fairly wide distribution.

There are 8 males and 13 females in the sample.

4 out of 21 volunteers reported having an upper-limb movement limitation, which is roughly 19% of the sample.

- Usability Evaluation

All questions in this section were marked with a score from 1 to 5.

Table 3: Usability questions average score (out of 5)

Question	**Average**
U1 (Overall Usability):	**4,84**
U2 (Ease of Exercises):	**4,69**
U3 (Technical Issues):	**3,99**
U4 (Intuitiveness):	**4,36**
U5 (Configuration Information Clarity):	**3,86**
U6 (Response Time):	**4,21**

U1 (Overall Usability): The average rating for overall usability is 4.81 (out of 5). This suggests that users generally found the app to be very easy to use.

U2 (Ease of Exercises): The average rating is 4.69. Users generally found the exercises easy to do.

U3 (Technical Issues): The average rating is 3,99. Most users did not encounter significant technical issues, though a few reported some problems. There is room for improvement here.

U4 (Intuitiveness): The average rating is 4,36. This suggests that the app is generally perceived as intuitive to use.

U5 (Configuration Information Clarity): The average rating is 3,86. This is relatively lower compared to other usability metrics, suggesting that the clarity of the configuration information might be an area for improvement.

U6 (Response Time): The average rating is 4,32 This suggests that most users found the app's response time to be fast.

- Musical Instrument Substitution

MI1 (Potential to Substitute a Traditional Instrument): Most of the volunteers (18 out of 21) either 'Agree' or 'Slightly Agree' that the app has the potential to substitute a traditional musical instrument in the context of music therapy for stroke recovery.

MI2 (Comparison to Playing a Traditional Percussive Instrument): The majority of volunteers (18 out of 21) either 'Agree' or 'Slightly Agree' that the use of the app is comparable to playing a traditional musical instrument. However, three volunteers 'Slightly Disagree' with this statement.

MI3 (Easier with a Musical Instrument): All volunteers (21 out of 21) answered 'NO' to whether they think the exercises would be easier to do with a musical instrument. This suggest that the users don't see a traditional instrument as necessary for the exercises.

5. Discussion

In this thesis, a significant consideration has been the selection of an appropriate machine learning framework for body landmark detection, critical for the accurate classification of post-stroke upper-limb motor activities. The majority of existing models for posture recognition, such as MoveNet, are designed with a focus on full-body perspectives and are typically trained on datasets of individuals in standing positions or doing full-body movements. This paradigm presents a limitation for applications like the one in this thesis, which targets populations that are typically in a seated position.

When experimenting with MoveNet, the body landmarks detected were misaligned, probably due to the seated position of the users. A classifier's effectiveness is intrinsically tied to the quality of its input data, and in this context, MoveNet was not sufficient. In contrast, with CoreML and Vision from Apple, body landmarks were more accurate for seated individuals. Contrary to MoveNet which shares the datasets used for this training, Vision does not specify which database has been used for its training as it is a commercial model. By seeing the results, it is more than likely that the database that they are using includes more seated examples. The decision of using CoreML models for this thesis enabled the development of the classification for the Fugl-Meyer Assessment activities in the resulting app.
This work, therefore, not only contributes a potential solution for post-stroke motor recovery with music therapy but also sheds light on the significant gap in existing machine learning models regarding accurate assessment of seated individuals, a common scenario in clinical evaluations.

Regarding the evaluation of the app, the results predominantly indicate a positive user experience. The usability questions indicate that users found the exercises easy to perform (4,69/5) and the app to be intuitive (4,36/5) with a satisfactory response time (4,21/5).
There are two areas that received a lower score. The clarity of the configuration information (3,86/5) indicates that users may benefit from more explicit instructions or guidance within the app. On the other hand, some users experienced technical issues

(3,99/5) with the detection of exercises. Users who were wearing flowing clothing reduced the model's accuracy when trying to detect their body key points, this is stated as a model limitation in Apple developer's Vision framework official webpage. The generated model for this thesis is also not perfect and might mistake an action for another in some cases.
A majority of the participants felt the app has the potential to substitute a traditional musical instrument, aligning with the goal of the application. Furthermore, all participants, regardless of their upper-limb mobility status, did not perceive the need for conventional instruments to facilitate the exercises and emphasized the independent value of the app.

There were certain inherent limitations when the topic of this work was decided. First, obviously, the constraints of using a model that can be deployed in a mobile application, as the app was the final goal. Also due to materials availability and that it is not possible to use an app simulator on the computer to test and debug an app that is using live video feed, the app was developed for an iOS environment. This study assumes that patients have access to and familiarity with the necessary technology, which could be a significant limitation among older adults. In order to prevent this issue, the app design is as simple as possible, assuming the user will be able to locate their mobile phone accordingly to the instructions. Also, due to time limitations, the study did not evaluate the long-term effects and retention of motor skills developed through the use of the app, which would help assess the sustained impact of this intervention.

If found to be effective, this research suggests that the app could be integrated into standard post-stroke rehabilitation protocols, thereby offering a novel, cost-effective, and engaging method for patients to improve their upper-limb motor function. The app's potential to make rehabilitation exercises more accessible could be transformative, especially for individuals who are unable to attend frequent in-person therapy sessions due to geographical, financial, or health-related constraints. This study also opens the door to more personalized, patient-centred care. The app's exercises could be adapted on each patient's progress and preferences. By integrating music therapy principles into

digital health interventions, this app could contribute to expanding the scope and acceptance of music therapy techniques in various clinical settings.

Future work in this field should consider a wider participant sample size, with a diverse demographic representation of the stroke survivor population. Participants with varying degrees of upper-limb motor impairment should be involved to assess the app's effectiveness across different stages of recovery, long-term. In addition to this, the integration with alternative machine learning models for detecting body landmarks in seated individuals should be investigated. As the current study uses CoreML and Vision from Apple, it may be beneficial to explore the development of platform-independent models that are accessible for various operation systems and devices.

6. Conclusions

This thesis demonstrates the potential for a mobile application, grounded in music therapy principles and utilizing machine learning techniques, to play a meaningful role in the upper-limb motor recovery of stroke survivors. The study found that participants generally rated the app as highly usable, with particularly positive feedback regarding its intuitive design and ease of exercises. Users also agree that the app has the potential to substitute a traditional percussive instrument in the context of music therapy for post-stroke rehabilitation. The chosen machine learning framework for body landmark detection and pose estimation is CoreML, combined with Vision from Apple, which proved to be more accurate for seated positions compared to other tested frameworks such as MoveNet. This work also sheds light on the gap that there is in machine learning models regarding proper assessment of body pose detection in seated individuals.

This research adds valuable insights to the field that merges music therapy, digital health technologies, and stroke recovery interventions. The results indicate that an application like the one developed in this thesis could be of great value to assist in the post-stroke recovery process, making rehabilitation more accessible and engaging for patients.

7. List of figures

8. List of tables

9. Bibliography

Ang, K. K., Guan, C., Phua, K. S., Wang, C., Zhou, L., Tang, K. Y., Ephraim Joseph, G. J., Kuah, C. W. K., & Chua, K. S. G. (2014). Brain-computer interface-based robotic end effector system for wrist and hand rehabilitation: Results of a three-armed randomized controlled trial for chronic stroke. *Frontiers in Neuroengineering*, *7*, 30. https://doi.org/10.3389/fneng.2014.00030

Braun Janzen, T., Koshimori, Y., Richard, N. M., & Thaut, M. H. (2022). Rhythm and Music-Based Interventions in Motor Rehabilitation: Current Evidence and Future Perspectives. *Frontiers in Human Neuroscience*, *15*. https://www.frontiersin.org/articles/10.3389/fnhum.2021.789467

Cervera, M. A., Soekadar, S. R., Ushiba, J., Millán, J. D. R., Liu, M., Birbaumer, N., & Garipelli, G. (2018). Brain-computer interfaces for post-stroke motor rehabilitation: A meta-analysis. *Annals of Clinical and Translational Neurology*, *5*(5), 651–663. https://doi.org/10.1002/acn3.544

Dodakian, L., McKenzie, A. L., Le, V., See, J., Pearson-Fuhrhop, K., Burke Quinlan, E., Zhou, R. J., Augsberger, R., Tran, X. A., Friedman, N., Reinkensmeyer, D. J., & Cramer, S. C. (2017). A Home-Based Telerehabilitation Program for Patients With Stroke. *Neurorehabilitation and Neural Repair*, *31*(10–11), 923–933. https://doi.org/10.1177/1545968317733818

Ganesan, V. (2022). Machine Learning in Mobile Applications. *International Journal of Computer Science and Mobile Computing*, *11*(2), 110–118. https://doi.org/10.47760/ijcsmc.2022.v11i02.013

Gladstone, D. J., Danells, C. J., & Black, S. E. (2002). The Fugl-Meyer Assessment of Motor Recovery after Stroke: A Critical Review of Its Measurement Properties. *Neurorehabilitation and Neural Repair*, *16*(3), 232–240. https://doi.org/10.1177/154596802401105171

Hesse, S., Schmidt, H., Werner, C., & Bardeleben, A. (2003). Upper and lower extremity robotic devices for rehabilitation and for studying motor control. *Current Opinion in Neurology*, *16*(6), 705–710. https://doi.org/10.1097/01.wco.0000102630.16692.38

Huang, W.-H., Dou, Z.-L., Jin, H.-M., Cui, Y., Li, X., & Zeng, Q. (2021). The Effectiveness of Music Therapy on Hand Function in Patients With Stroke: A Systematic Review of Randomized Controlled Trials. *Frontiers in Neurology*, *12*. https://www.frontiersin.org/articles/10.3389/fneur.2021.641023

Laver, K. E., Lange, B., George, S., Deutsch, J. E., Saposnik, G., & Crotty, M. (2017). Virtual reality for stroke rehabilitation. *The Cochrane Database of Systematic Reviews*, *11*(11), CD008349. https://doi.org/10.1002/14651858.CD008349.pub4

Lohse, K. R., Hilderman, C. G. E., Cheung, K. L., Tatla, S., & Van der Loos, H. F. M. (2014). Virtual Reality Therapy for Adults Post-Stroke: A Systematic Review and Meta-Analysis Exploring Virtual Environments and Commercial Games in Therapy. *PLoS ONE*, *9*(3), e93318. https://doi.org/10.1371/journal.pone.0093318

Mehrholz, J., Pohl, M., Platz, T., Kugler, J., & Elsner, B. (2015). Electromechanical and robot-assisted arm training for improving activities of daily living, arm function, and arm muscle strength after stroke. *The Cochrane Database of Systematic Reviews*, *2015*(11), CD006876. https://doi.org/10.1002/14651858.CD006876.pub4

Porciuncula, F., Roto, A. V., Kumar, D., Davis, I., Roy, S., Walsh, C. J., & Awad, L. N. (2018). Wearable Movement Sensors for Rehabilitation: A Focused Review of Technological and Clinical Advances. *PM & R : The Journal of Injury, Function, and Rehabilitation*, *10*(9 Suppl 2), S220–S232. https://doi.org/10.1016/j.pmrj.2018.06.013

Rodriguez-Prunotto, L., & Cano-de-la-Cuerda, R. (2018). [Mobile applications related to stroke: A systematic review]. *Revista De Neurologia*, *66*(7), 213–229.

Saposnik, G., Cohen, L. G., Mamdani, M., Pooyania, S., Ploughman, M., Cheung, D., Shaw, J., Hall, J., Nord, P., Dukelow, S., Nilanont, Y., De Los Rios, F., Olmos, L., Levin, M., Teasell, R., Cohen, A., Thorpe, K., Laupacis, A., Bayley, M., & Stroke Outcomes Research Canada. (2016). Efficacy and safety of non-immersive virtual reality exercising in stroke rehabilitation (EVREST): A randomised, multicentre, single-blind, controlled trial. *The Lancet. Neurology*, *15*(10), 1019–1027. https://doi.org/10.1016/S1474-4422(16)30121-1

Sawant, N., Bose, M., & Parab, S. (2020). Dexteria app. Therapy versus conventional hand therapy in stroke. *Journal of Enabling Technologies*, *14*(4), 221–231. https://doi.org/10.1108/JET-05-2020-0023

Segura, E., Grau-Sánchez, J., Sanchez-Pinsach, D., De la Cruz, M., Duarte, E., Arcos, J. L., & Rodríguez-Fornells, A. (2021). Designing an app for home-based enriched Music-supported Therapy in the rehabilitation of patients with chronic stroke: A pilot feasibility study. *Brain Injury*, *35*(12–13), 1585–1597. https://doi.org/10.1080/02699052.2021.1975819

Tchero, H., Tabue Teguo, M., Lannuzel, A., & Rusch, E. (2018). Telerehabilitation for Stroke Survivors: Systematic Review and Meta-Analysis. *Journal of Medical Internet Research*, *20*(10), e10867. https://doi.org/10.2196/10867

Xu, C., He, Z., Shen, Z., & Huang, F. (2022). Potential Benefits of Music Therapy on Stroke Rehabilitation. *Oxidative Medicine and Cellular Longevity*, *2022*, 9386095. https://doi.org/10.1155/2022/9386095

A. Appendices

a. README file

FuglMeyerApp is an iOS application that utilizes computer vision and audio playback to detect specific body movements included in the Fugl Meyer assessment for stroke recovery, and provide audio feedback when actions are completed. The app captures video from the device's front camera, processes it to estimate body positions, and plays corresponding audio sounds based on the detected movements.

The codebase consists of three main components:

1. **ViewController.swift**: This file contains the main view controller class responsible for managing the app's user interface and coordinating interactions with the video capture and prediction components.
2. 2. **Predictor.swift**: The Predictor class is responsible for analyzing the body pose observations received from the video capture and making predictions about the detected actions. It utilizes a trained machine learning model (*FMClassifier*) to label the actions and calculates the confidence level for each prediction. It also delegates the recognized points to the ViewController for visualization.
3. 3. **VideoCapture.swift**: The VideoCapture class handles the capture of video data from the device's front camera. It configures an AVCaptureSession to manage the input (camera) and output (video data) and sets up an AVCaptureVideoDataOutput to receive the video frames. It also delegates the received sample buffers to the Predictor for pose estimation.

##Dependencies
- AVFoundation: Used for capturing video data, audio playback, and managing AVAudioPlayer instances.
- Vision: Provides support for body pose estimation using VNHumanBodyPoseObservation.
- CoreML: Enables the integration and utilization of a trained machine learning model (FMClassifier) for action prediction.

##Getting Started

To run the FuglMeyerApp, follow these steps:

1. Clone the repository to your local machine.
2. Open the project in Xcode.
3. Build and run the app on your iOS device or simulator.
4. Grant permission for the app to access the camera.

##Behavior

1. When the app is launched, the ViewController's `viewDidLoad()`method is called, triggering the setup of the video preview and audio file preloading. The AVCaptureVideoPreviewLayer is added as a sublayer to the view, allowing real-time video display. Audio files for ear and lateral sounds are preloaded using AVAudioPlayer.

2. The `setupVideoPreview()`method initializes the video capture by starting the capture session and setting up the AVCaptureVideoPreviewLayer for previewing the captured video.

3. The `preloadAudioFiles()`method loads the ear and lateral sound files into AVAudioPlayer instances to be played later during action detection.

4. The ViewController conforms to the PredictorDelegate protocol, implementing the delegate methods `didFindNewRecognizedPoints()`and `didLabelAction()`. These methods are called by the Predictor when new recognized body points or labeled actions are available.

5. The Predictor class estimates body poses by receiving CMSampleBuffer objects through the `estimation(sampleBuffer:)`method. It uses Vision and Core ML to process the video frames and extract VNHumanBodyPoseObservation instances.

6. For each received VNHumanBodyPoseObservation, the recognized body points are extracted and transformed to display coordinates. The transformed points are then passed to the ViewController using the delegate method `didFindNewRecognizedPoints()`.

7. The Predictor maintains a window of VNHumanBodyPoseObservations for prediction purposes. When a new observation is received, it is added to the window, and the oldest observation is removed if the window reaches its maximum size (predictionWindowSize).

8. The Predictor uses a trained machine learning model (FMClassifier) to label the actions based on the stored observations in the window. The labeled action and its confidence level are sent to the ViewController using the delegate method `didLabelAction()`.

9. When an action is labeled and its confidence level exceeds the specified threshold, the ViewController triggers the corresponding behavior. If the action is "ear," the ear sound is played using the AVAudioPlayer for auditory feedback. If the action is "90lateral," the lateral sound is played.

10. A delay of 3 seconds is introduced after an action is detected to prevent rapid and repeated action triggering. The corresponding flag (isClapDetected or isArmDetected) is reset after the delay to allow the detection of subsequent actions.

b. Appendix – Usability questionnaire

1. General information:

Age:

Gender:

Upper-limb restrained mobility (yes/no):

2. Usability Evaluation:

U1: On a scale of 1 to 5, rate the overall usability of the app, with 1 being very difficult to use and 5 very easy to use.

U2: On a scale of 1 to 5, rate how easy to do were the exercises, with 1 being very difficult and 5 very easy.

U3: On a scale of 1 to 5, rate the behaviour of the app, with 1 being that there were a lot of technical issues or bugs and 5 no technical issues or bugs.

U4: On a scale of 1 to 5, rate how intuitive was the app to use, with 1 being not intuitive at all, 5 very intuitive.

U5: On a scale of 1 to 5, rate the configuration information presented, with 1 being very poor information and 5 very clear information.

U6: On a scale of 1 to 5, how fast was the app's response, with 1 being really slow and 5 very fast.

3. Musical instrument Substitution:

MI1: Agree, Slightly Agree, Slightly Disagree, Disagree. The app has the potential to substitute a traditional musical instrument.

MI2: Agree, Slightly Agree, Slightly Disagree, Disagree. The use of the app is comparable to playing a traditional musical instrument.

MI3: Do you think the exercises would be easier to do with a musical instrument?

c. Appendix – Usability Questionnaire Results

Table 4: Questionnaire Results Table

RESULTS																					
Volunteer Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Age	60	24	63	83	40	36	70	28	66	85	45	38	52	30	29	32	68	48	41	55	35
Gender	F	M	M	F	F	F	F	M	F	F	M	F	M	M	M	M	F	M	F	M	M
Upper-limb limitation	YES	NO	NO	NO	NO	NO	YES	NO	YES	NO	NO	YES	NO	NO	NO	NO	YES	NO	NO	NO	NO
U1	5	5	5	4	5	5	4	5	5	5	5	5	5	5	5	5	4	5	5	5	5
U2	4	5	5	5	5	5	4	5	4	5	5	4	5	5	5	5	4	5	4	5	5
U3	5	4	4	5	5	1	4	5	3	5	5	3	5	5	5	5	3	5	2	5	5
U4	5	5	5	4	4	3	4	5	4	4	5	3	5	5	5	5	4	5	3	5	5
U5	4	3	3	3	5	5	3	4	3	5	3	4	5	3	4	4	3	5	4	5	5
U6	5	4	4	5	3	3	4	5	4	5	4	3	5	5	5	5	4	4	3	5	5
MI1	Slightly Agree	Slightly Agree	Agree	Slightly Agree	Agree	Agree	Slightly Agree	Agree	Agree	Slightly Agree	Slightly Agree	Slightly Agree	Agree	Agree	Slightly Agree	Agree	Agree	Slightly Agree	Agree	Agree	Agree
MI2	Slightly Agree	Slightly Agree	Slightly Disagree	Slightly Agree	Agree	Slightly Agree	Slightly Agree	Agree	Slightly Disagree	Slightly Agree	Agree	Slightly Agree	Slightly Agree	Agree	Agree	Agree	Slightly Disagree	Slightly Agree	Slightly Agree	Slightly Agree	Agree
MI3	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

d. Appendix – MoveNet screen captures code

```
import tensorflow as tf

import tensorflow_hub as hub

import cv2

import os


# Load the MoveNet "thunder" model from TensorFlow Hub

#model_url = "https://tfhub.dev/google/movenet/singlepose/thunder/1"

model_url = "https://tfhub.dev/google/movenet/singlepose/lightning/4"

model = hub.load(model_url)


def movenet(input_image):

    input_tensor = tf.image.convert_image_dtype(input_image,

dtype=tf.int32)[tf.newaxis, ...]

    keypoints_with_scores = model.signatures['serving_default'](input_tensor)

    return keypoints_with_scores['output_0']


def process_video(video_path):

    cap = cv2.VideoCapture(video_path)
```

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # Convert the BGR image to RGB
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    #for thunder:
    #frame_rgb = cv2.resize(frame_rgb, (256,256))
    #for lightning:
    frame_rgb = cv2.resize(frame_rgb, (192,192))
    # Predict the pose
    keypoints_with_scores = movenet(frame_rgb)
    keypoints = keypoints_with_scores[0].numpy()[0, :, :2]


    # Visualize the keypoints on the image
    for kp in keypoints:
        y, x = kp
```

```
            cv2.circle(frame, (int(x * frame.shape[1]), int(y * frame.shape[0])), 3, (0,
255, 0), -1)


        cv2.imshow("MoveNet Pose Estimation", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()


# Assuming your dataset is organized as "base_dir/ClassName/video_file.mp4"
base_dir = "/Users/mireiadegracia/Documents/UNI/Fugl Recovery
2/new_dataset_clips"
for class_folder in os.listdir(base_dir):
    if class_folder != '.DS_Store':

        class_path = os.path.join(base_dir, class_folder)

        for video_file in os.listdir(class_path):
            if video_file != '.DS_Store':
```

```
video_path = os.path.join(class_path, video_file)

process_video(video_path)
```

e. Appendix – Main APP Code Parts

Predictor

```
//
//  Predictor.swift
//  FuglMeyerApp
//
//  Created by Mireia de Gracia on 18/5/23.
//

import Foundation
import Vision

// typealias FMClassifier = fugl_meyer_test_1
//typealias FMClassifier = MyActionClassifier_2
//typealias FMClassifier = fugl_meyer_test_12
//typealias FMClassifier = FuglMeyerActionClassifier_1_Iteration_40
typealias FMClassifier = FuglMeyerActionClassifier_3_Iteration_60
//typealias FMClassifier = Fugl_meyer_test_9_Iteration_40
// typealias FMClassifier = Fugl_meyer_classifier_2

protocol PredictorDelegate: AnyObject {
   func predictor(_ predictor: Predictor, didFindNewRecognizedPoints points:
[CGPoint])
   func predictor(_ predictor: Predictor, didLabelAction action: String, with
confidence: Double)
}

class Predictor {
```

```
    let predictionWindowSize = 30 // Number of observations to keep in the
posesWindow
    var posesWindow: [VNHumanBodyPoseObservation] = [] // Window of
VNHumanBodyPoseObservation objects

    init() {
        posesWindow.reserveCapacity(predictionWindowSize) // Reserve capacity
for the posesWindow array
    }

    weak var delegate: PredictorDelegate? // Delegate to notify about recognized
points and labeled actions

    func estimation(sampleBuffer: CMSampleBuffer) {
        let requestHandler = VNImageRequestHandler(cmSampleBuffer:
sampleBuffer, orientation: .up)

        let request = VNDetectHumanBodyPoseRequest(completionHandler:
bodyPoseHandler)

        do {
            try requestHandler.perform([request]) // Perform the request to detect
human body pose
        } catch {
            print("Unable to perform the request with error: \(error)")
        }
    }
```

```
    func prepareInputWithObservations(_ observations:
[VNHumanBodyPoseObservation]) -> MLMultiArray? {
        let numAvailableFrames = observations.count
        let observationsNeeded = 30
        var multiArrayBuffer = [MLMultiArray]()

        for frameIndex in 0 ..< min(numAvailableFrames, observationsNeeded) {
            let pose = observations[frameIndex]
            do {
                let oneFrameMultiarray = try pose.keypointsMultiArray() // Get the
keypoints as a multi-array
                multiArrayBuffer.append(oneFrameMultiarray) // Append the multi-
array to the buffer
            } catch {
                continue
            }
        }

        if numAvailableFrames < observationsNeeded {
            for _ in 0 ..< (observationsNeeded - numAvailableFrames) {
                do {
                    let oneFrameMultiArray = try MLMultiArray(shape: [1, 3, 18],
dataType: .double)
                    multiArrayBuffer.append(oneFrameMultiArray)
                    try resetMultiArray(oneFrameMultiArray)
                } catch {
                    continue
                }
```

```
            }
        }

        return MLMultiArray(concatenating: [MLMultiArray](multiArrayBuffer), axis:
0, dataType: .float) // Concatenate the multi-arrays along axis 0
    }

    func resetMultiArray(_ predictionWindow: MLMultiArray, with value: Double =
0.0) throws {
        let pointer = try UnsafeMutableBufferPointer<Double>(predictionWindow)
        pointer.initialize(repeating: value) // Set all values in the multi-array to the
given value
    }

    func bodyPoseHandler(request: VNRequest, error: Error?) {
        guard let observations = request.results as?
[VNHumanBodyPoseObservation] else { return }

        observations.forEach {
            processObservation($0) // Process each observation and notify the
delegate about recognized points
        }

        if let result = observations.first {
            storeObservation(result) // Store the first observation in the
posesWindow
```

```
            labelActionType() // Label the action type based on the stored
observations
        }
    }

    func labelActionType() {
        guard let movementsClassifier = try? FMClassifier(configuration:
MLModelConfiguration()),
              let poseMultiArray = prepareInputWithObservations(posesWindow),
              let predictions = try? movementsClassifier.prediction(poses:
poseMultiArray) else {
            return
        }

        let label = predictions.label // Get the predicted action label
        let confidence = predictions.labelProbabilities[label] ?? 0 // Get the
confidence of the predicted action

        delegate?.predictor(self, didLabelAction: label, with: confidence) // Notify
the delegate about the labeled action
    }

    func storeObservation(_ observation: VNHumanBodyPoseObservation) {
        if posesWindow.count >= predictionWindowSize {
            posesWindow.removeFirst() // Remove the oldest observation if the
posesWindow is full
        }
```

```
        posesWindow.append(observation) // Append the new observation to the
posesWindow
    }

    func processObservation(_ observation: VNHumanBodyPoseObservation) {
        do {
            let recognizedPoints = try observation.recognizedPoints(forGroupKey:
.all) // Get the recognized points for all groups

            var displayedPoints = recognizedPoints.map {
                CGPoint(x: $0.value.x, y: 1 - $0.value.y) // Adjust the coordinates of
recognized points for display
            }
            delegate?.predictor(self, didFindNewRecognizedPoints:
displayedPoints) // Notify the delegate about the recognized points
        } catch {
            print("error finding recognizedPoints")
        }
    }
}
```

ViewController

```
//
//  ViewController.swift
//  FuglMeyerApp
//
//  Created by Mireia de Gracia on 17/5/23.
//

import UIKit
import AVFoundation
import AudioToolbox

class ViewController: UIViewController {
    var audioPlayer: AVAudioPlayer? // Used for playing ear sound
    var lateralAudioPlayer: AVAudioPlayer? // Used for playing lateral sound
    var earSoundURL: URL! // URL of the ear sound file
    var lateralSoundURL: URL! // URL of the lateral sound file

    // to preview some actual data
    let videoCapture = VideoCapture() // VideoCapture instance for capturing
video

    var previewLayer: AVCaptureVideoPreviewLayer? // Preview layer for
displaying captured video
    var isEarDetected = false // Flag to track if an ear movement is detected
    var isArmDetected = false // Flag to track if an arm movement is detected
    var pointsLayer = CAShapeLayer() // Layer for drawing recognized points

    private func setupVideoPreview() {
```

```
        videoCapture.startCaptureSession() // Start the video capture session

        previewLayer = AVCaptureVideoPreviewLayer(session:
videoCapture.captureSession) // Create preview layer

        do {
            try AVAudioSession.sharedInstance().setCategory(.playback) // Set
audio session category for playback
            try AVAudioSession.sharedInstance().setActive(true) // Activate the
audio session
        } catch {
            // Handle any errors that occur during audio session configuration
            print("Failed to configure audio session: \(error.localizedDescription)")
        }

        // Ensure that the previewLayer is initialized
        guard let previewLayer = previewLayer else { return }

        view.layer.addSublayer(previewLayer) // Add preview layer to the view's
layer
        previewLayer.frame = view.frame // Set the frame of the preview layer

        view.layer.addSublayer(pointsLayer) // Add points layer to the view's layer
        pointsLayer.frame = view.frame // Set the frame of the points layer
        pointsLayer.strokeColor = UIColor.green.cgColor // Set the stroke color of
the points layer
    }
```

```
    private func preloadAudioFiles() {
        // Preload ear sound
        if let earSoundPath = Bundle.main.path(forResource: "sound_test_ear",
ofType: "wav") {
            earSoundURL = URL(fileURLWithPath: earSoundPath)
            do {
                audioPlayer = try AVAudioPlayer(contentsOf: earSoundURL)
                audioPlayer?.prepareToPlay()
            } catch {
                print("Failed to preload ear sound: \(error.localizedDescription)")
            }
        }

        // Preload lateral sound
        if let lateralSoundPath = Bundle.main.path(forResource: "sound_test_arm",
ofType: "wav") {
            lateralSoundURL = URL(fileURLWithPath: lateralSoundPath)
            do {
                lateralAudioPlayer = try AVAudioPlayer(contentsOf:
lateralSoundURL)
                lateralAudioPlayer?.prepareToPlay()
            } catch {
                print("Failed to preload lateral sound: \(error.localizedDescription)")
            }
        }
    }

    override func viewDidLoad() {
```

```
        super.viewDidLoad()
        // Do any additional setup after loading the view.

        setupVideoPreview() // Set up the video preview

        videoCapture.predictor.delegate = self // Set the delegate for the video
capture's predictor

        preloadAudioFiles() // Preload audio files for playback
    }
}

extension ViewController: PredictorDelegate {
    func predictor(_ predictor: Predictor, didLabelAction action: String, with
confidence: Double) {
        // Action labeling delegate method
        print(confidence)
        print(action)
        if (action == "ear_right" || action == "ear_left") && confidence > 0.80 &&
!isEarDetected{
            // If the action is "ear" and the confidence level is high enough, and no
ear is currently detected

            //print("ear detected")
//            print(confidence)
            isEarDetected = true // Set clap detected flag

            DispatchQueue.main.async {
```

```
            if let audioPlayer = self.audioPlayer {
                audioPlayer.play() // Play the ear sound asynchronously
            }
        }

        DispatchQueue.main.asyncAfter(deadline: .now() + 4) {
            self.isEarDetected = false // Reset clap detected flag after a delay
        }
    }
    else if (action == "90lateral_right" || action == "90lateral_left") &&
confidence > 0.80 && !isArmDetected{
        // If the action is "90lateral" and the confidence level is high enough, and
no arm movement is currently detected

//          print("arm detected")
//          print(confidence)
        isArmDetected = true // Set arm detected flag

        DispatchQueue.main.async {
            if let lateralAudioPlayer = self.lateralAudioPlayer {
                lateralAudioPlayer.play() // Play the lateral sound asynchronously
            }
        }

        DispatchQueue.main.asyncAfter(deadline: .now() + 3) {
            self.isArmDetected = false // Reset arm detected flag after a delay
        }
    }
```

```
    }

    func predictor(_ predictor: Predictor, didFindNewRecognizedPoints points:
[CGPoint]) {
        // Point recognition delegate method

        guard let previewLayer = previewLayer else { return }

        let convertedPoints = points.map {
            previewLayer.layerPointConverted(fromCaptureDevicePoint: $0) //
Convert points to the layer's coordinate system
        }

        let combinedPath = CGMutablePath() // Create a combined path to draw
recognized points

        for point in convertedPoints {
            let dotPath = UIBezierPath(ovalIn: CGRect(x: point.x, y: point.y, width:
5, height: 5)) // Create an oval shape for each point
            combinedPath.addPath(dotPath.cgPath) // Add the oval shape to the
combined path
        }

        pointsLayer.path = combinedPath // Set the path of the points layer to the
combined path

        DispatchQueue.main.async {
```

```
            self.pointsLayer.didChangeValue(for: \.path) // Update the points layer
asynchronously to reflect the changes made
        }
    }
}
```

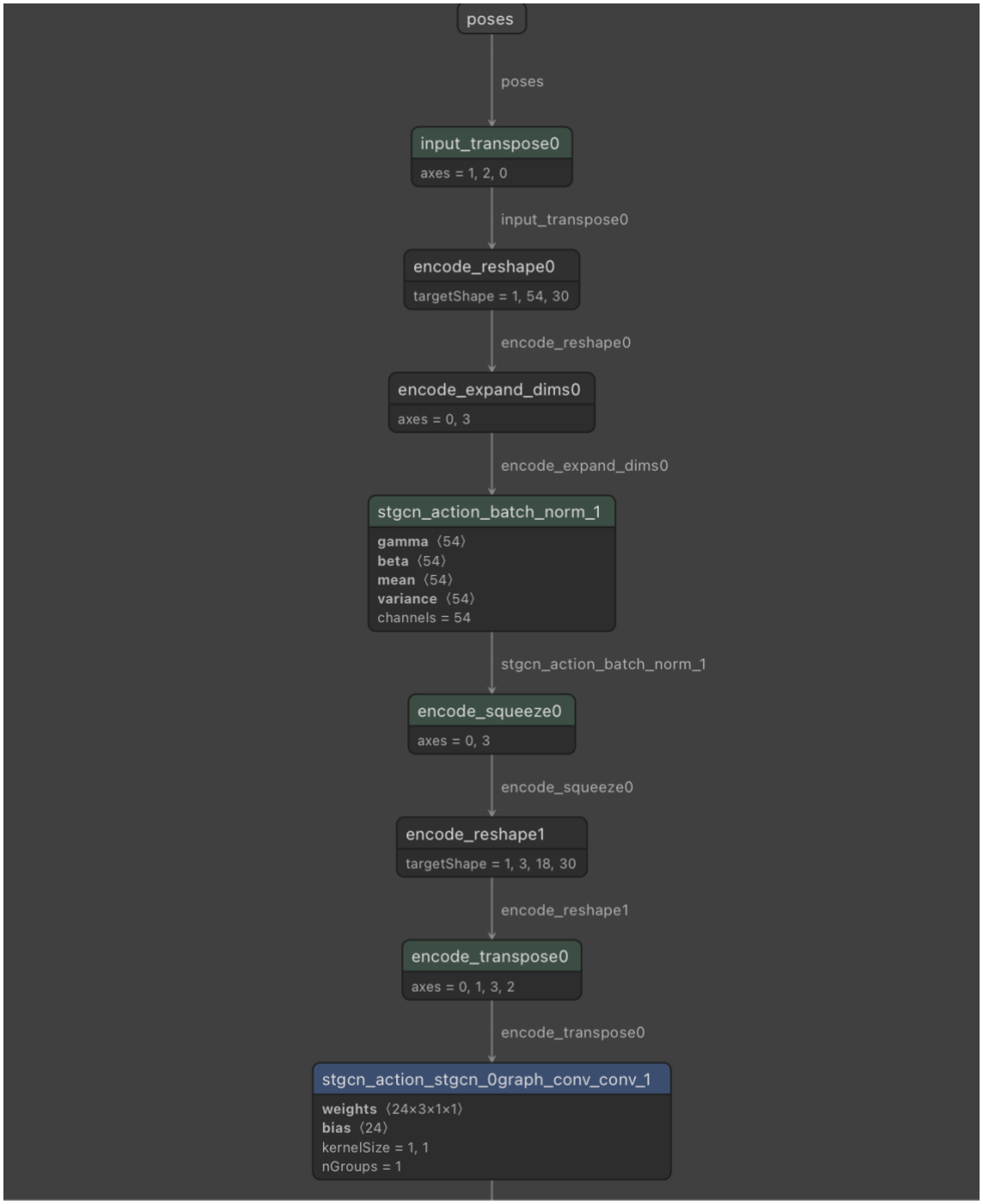
- **VideoCapture**

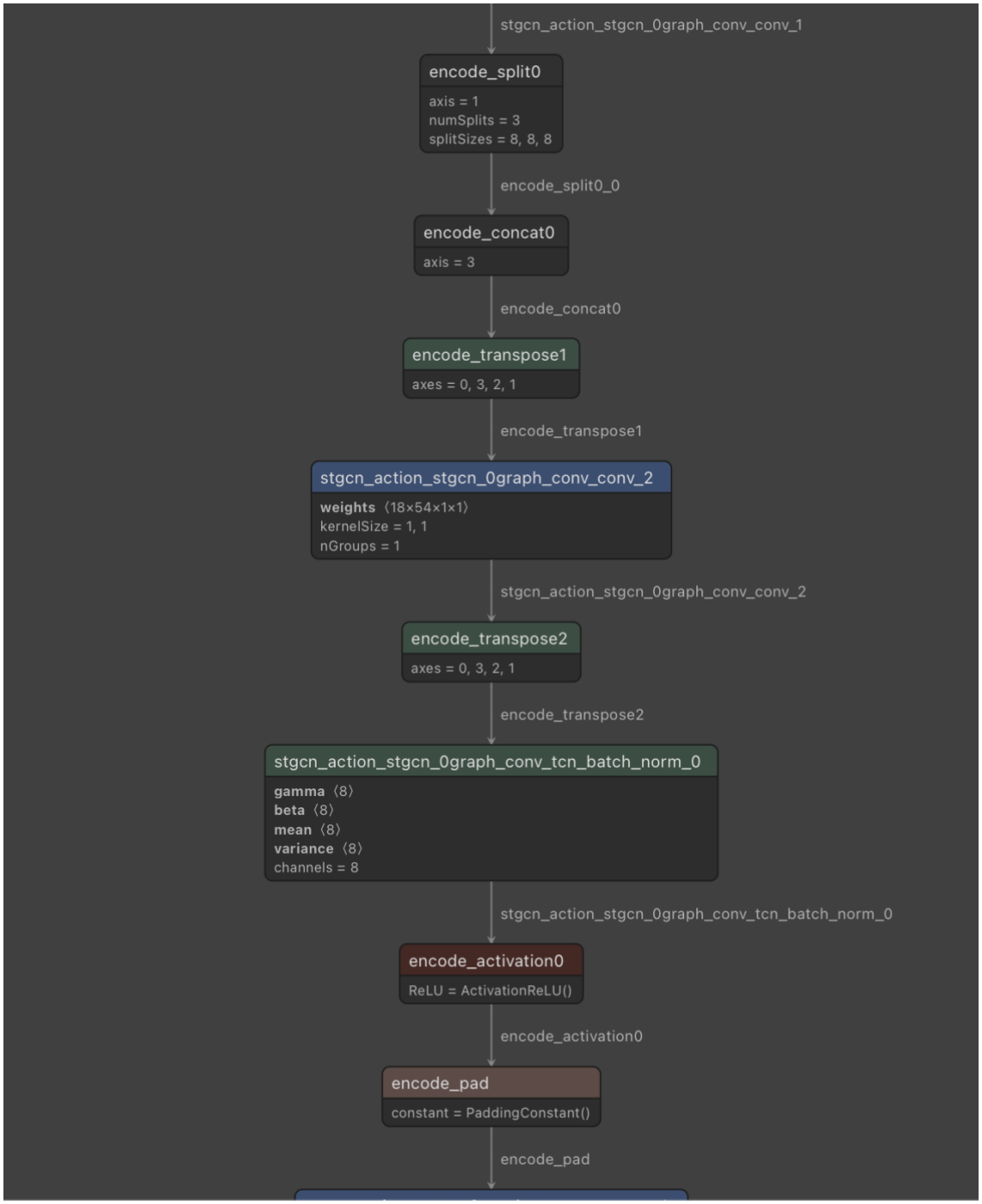
```
//
//  VideoCapture.swift
//  FuglMeyerApp
//
//  Created by Mireia de Gracia on 17/5/23.
//

import Foundation
import AVFoundation

class VideoCapture: NSObject {
    let captureSession = AVCaptureSession() // Capture session to manage input and output

    let videoOutput = AVCaptureVideoDataOutput() // Output to capture video data

    let predictor = Predictor() // Predictor for body position

    override init() {
        super.init()

        // Specify the capture device (change default camera to front camera!!!)
        // Create an AVCaptureDeviceInput to get data from the capture device
        guard let captureDevice = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInWideAngleCamera], mediaType: .video, position: .front).devices.first,
            let input = try? AVCaptureDeviceInput(device: captureDevice) else {
```

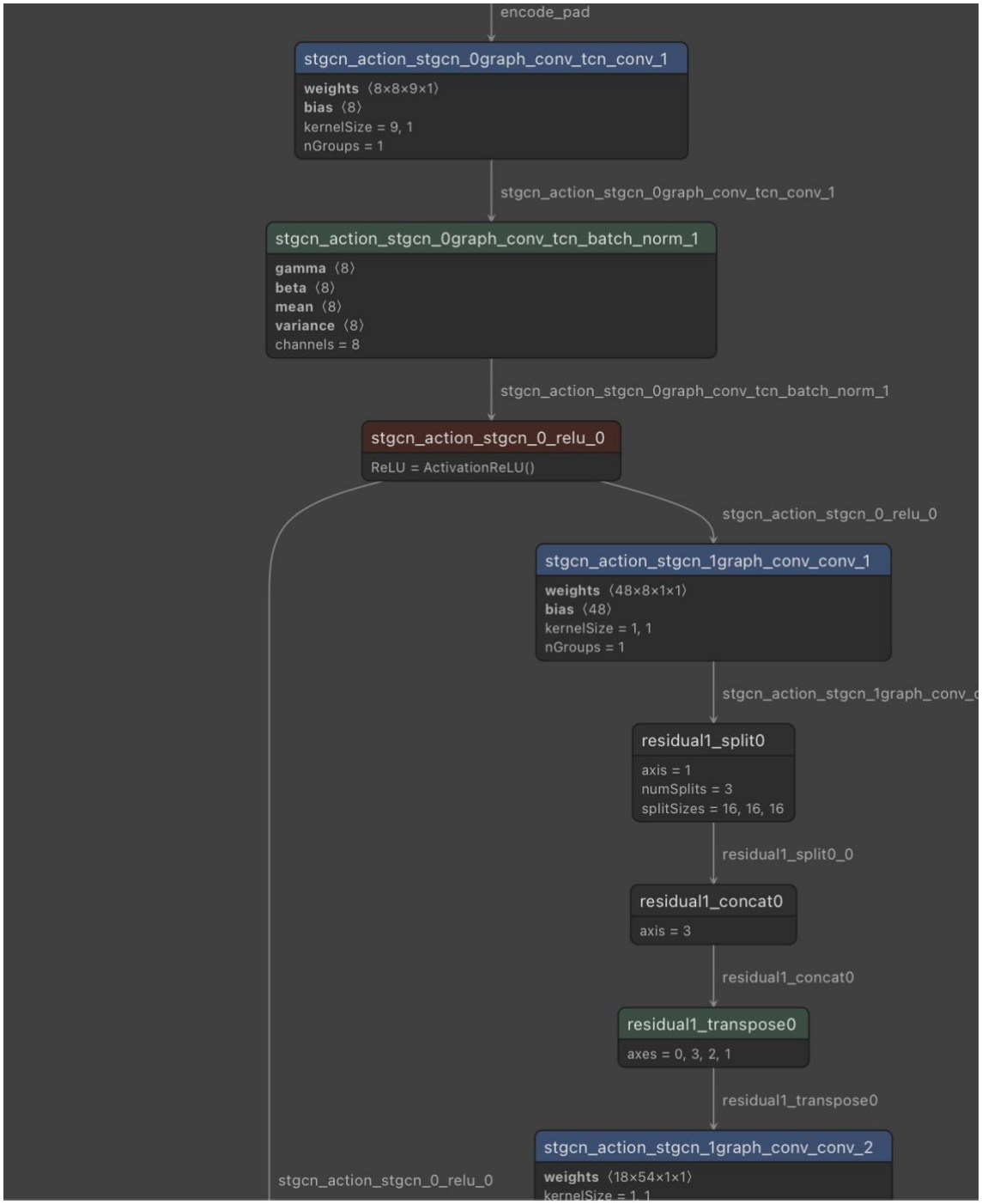
```
            return
        }

        // Configure the capture session
        captureSession.sessionPreset = AVCaptureSession.Preset.high // Set
data resolution to high
        captureSession.addInput(input) // Add the input to the capture session
        captureSession.addOutput(videoOutput) // Add the video output to the
capture session
        videoOutput.alwaysDiscardsLateVideoFrames = true // Discard late video
frames to reduce latency
    }

    func startCaptureSession() {
        captureSession.startRunning() // Start the capture session

        // Set the VideoCapture class as the delegate for video output
        videoOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label:
"videoDispatchQueue"))
    }
}

extension VideoCapture: AVCaptureVideoDataOutputSampleBufferDelegate {
    func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer:
CMSampleBuffer, from connection: AVCaptureConnection) {
        predictor.estimation(sampleBuffer: sampleBuffer) // Perform body position
estimation using the predictor
```

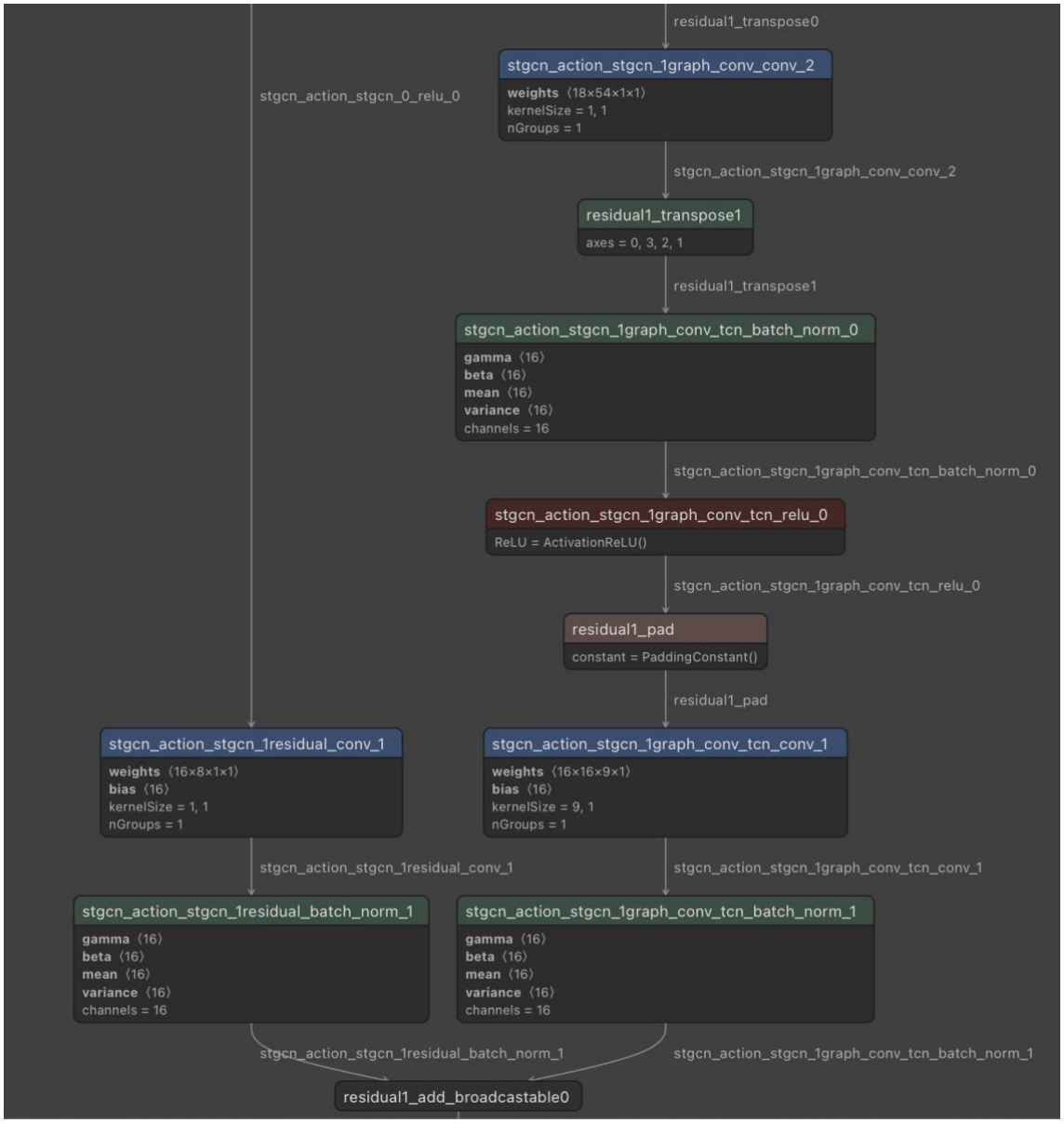
```
    // Additional code for processing the video data if needed
    // let videoData = sampleBuffer
    // print(videoData)
}
```

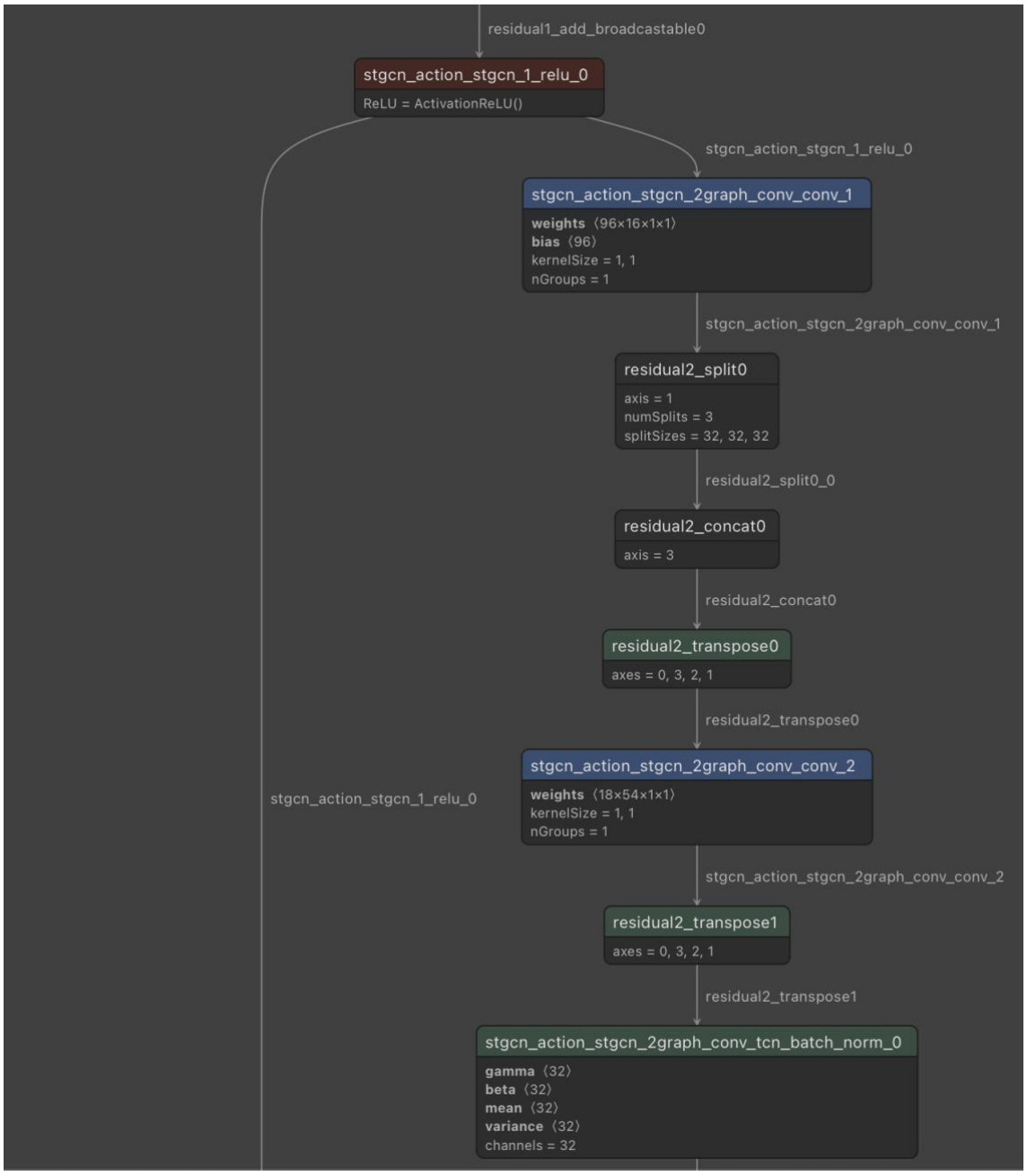
f. Appendix – Action Classifier CoreML trained model structure

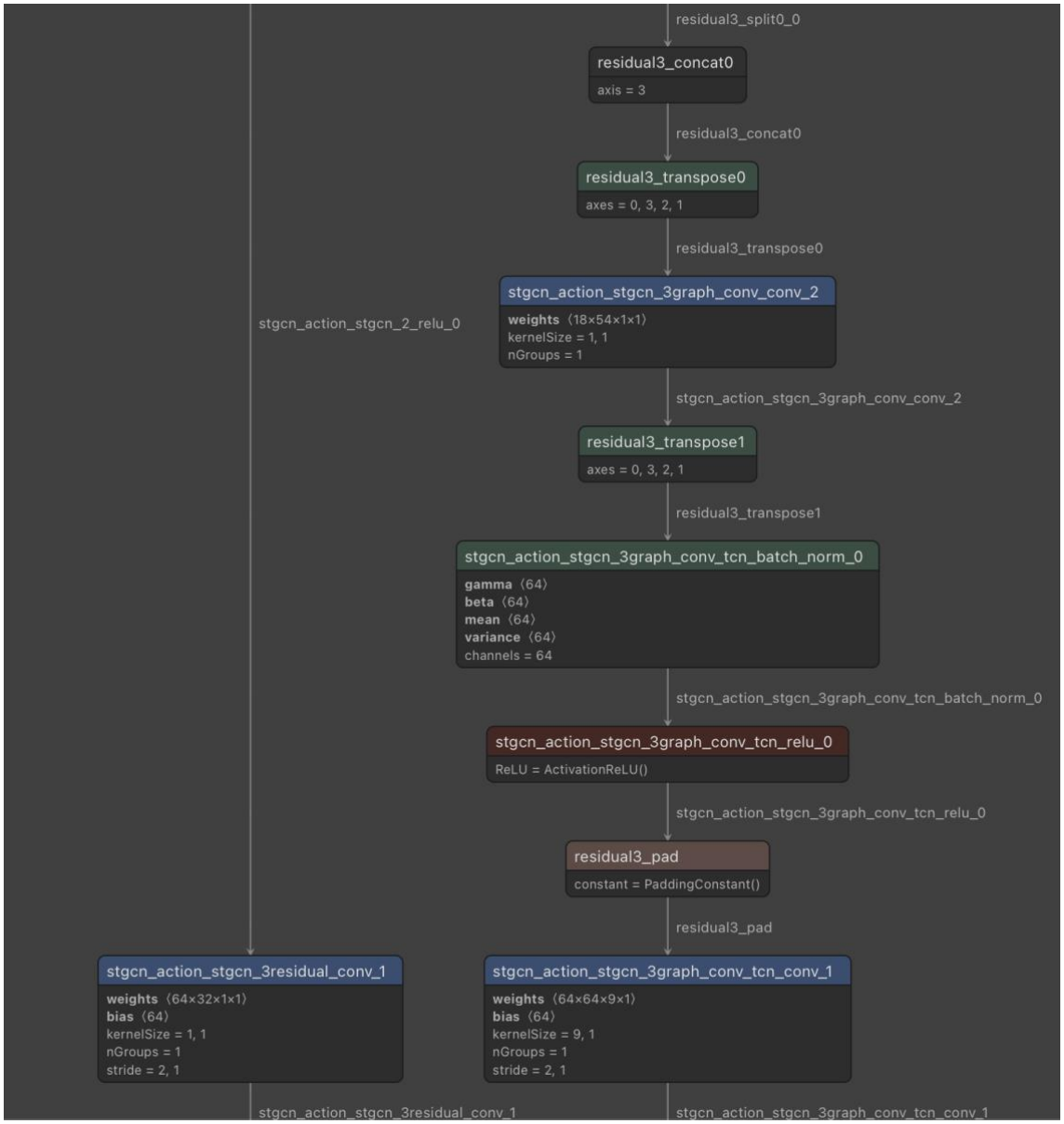


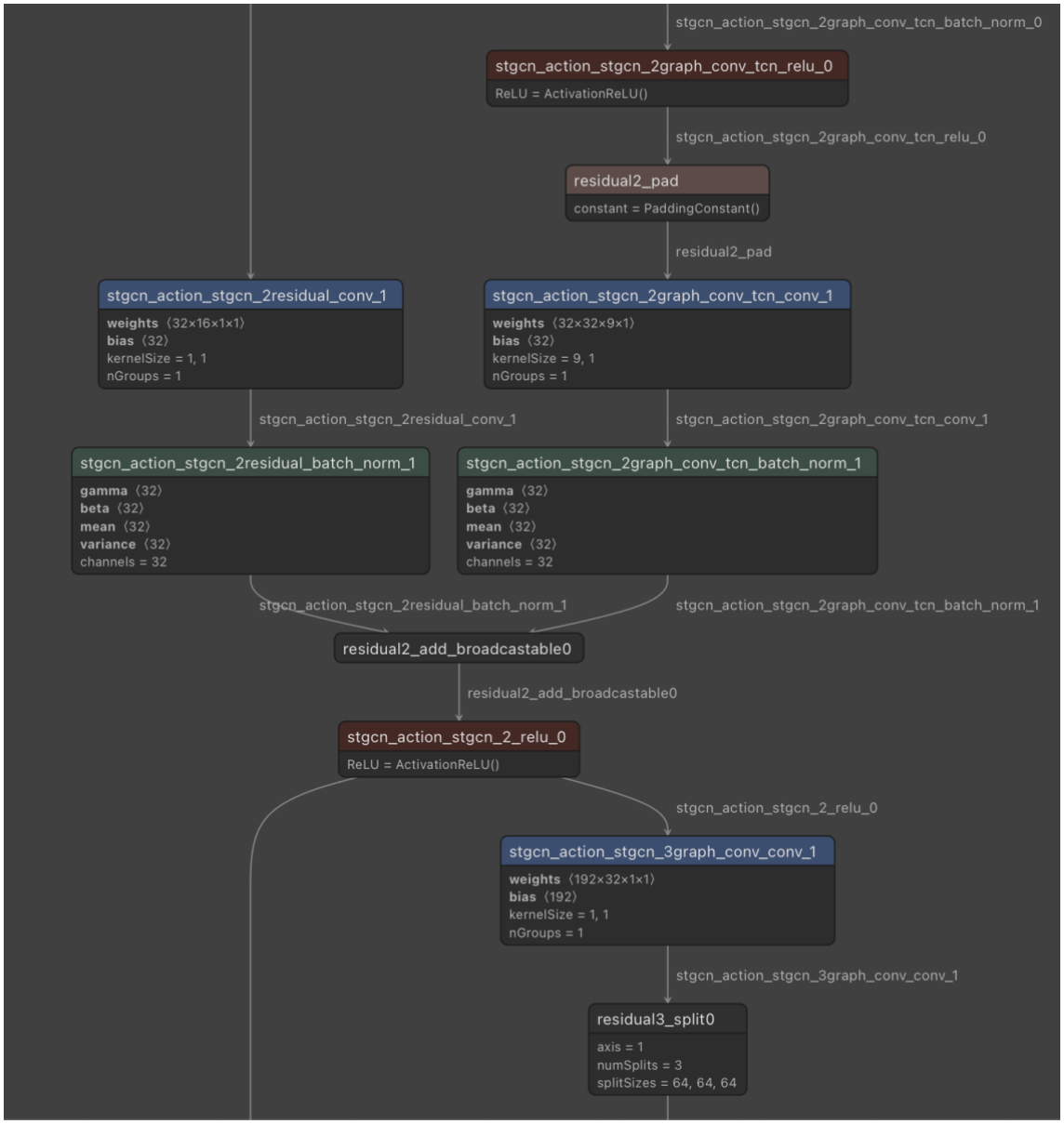

stgcn_action_stgcn_0graph_conv_conv_1
encode_split0
axis = 1
numSplits = 3
splitSizes = 8, 8, 8
encode_split0_0
encode_concat0
axis = 3
encode_concat0
encode_transpose1
axes = 0, 3, 2, 1
encode_transpose1
stgcn_action_stgcn_0graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_0graph_conv_conv_2
encode_transpose2
axes = 0, 3, 2, 1
encode_transpose2
stgcn_action_stgcn_0graph_conv_tcn_batch_norm_0
gamma ⟨8⟩
beta ⟨8⟩
mean ⟨8⟩
variance ⟨8⟩
channels = 8
stgcn_action_stgcn_0graph_conv_tcn_batch_norm_0
encode_activation0
ReLU = ActivationReLU()
encode_activation0
encode_pad
constant = PaddingConstant()
encode_pad

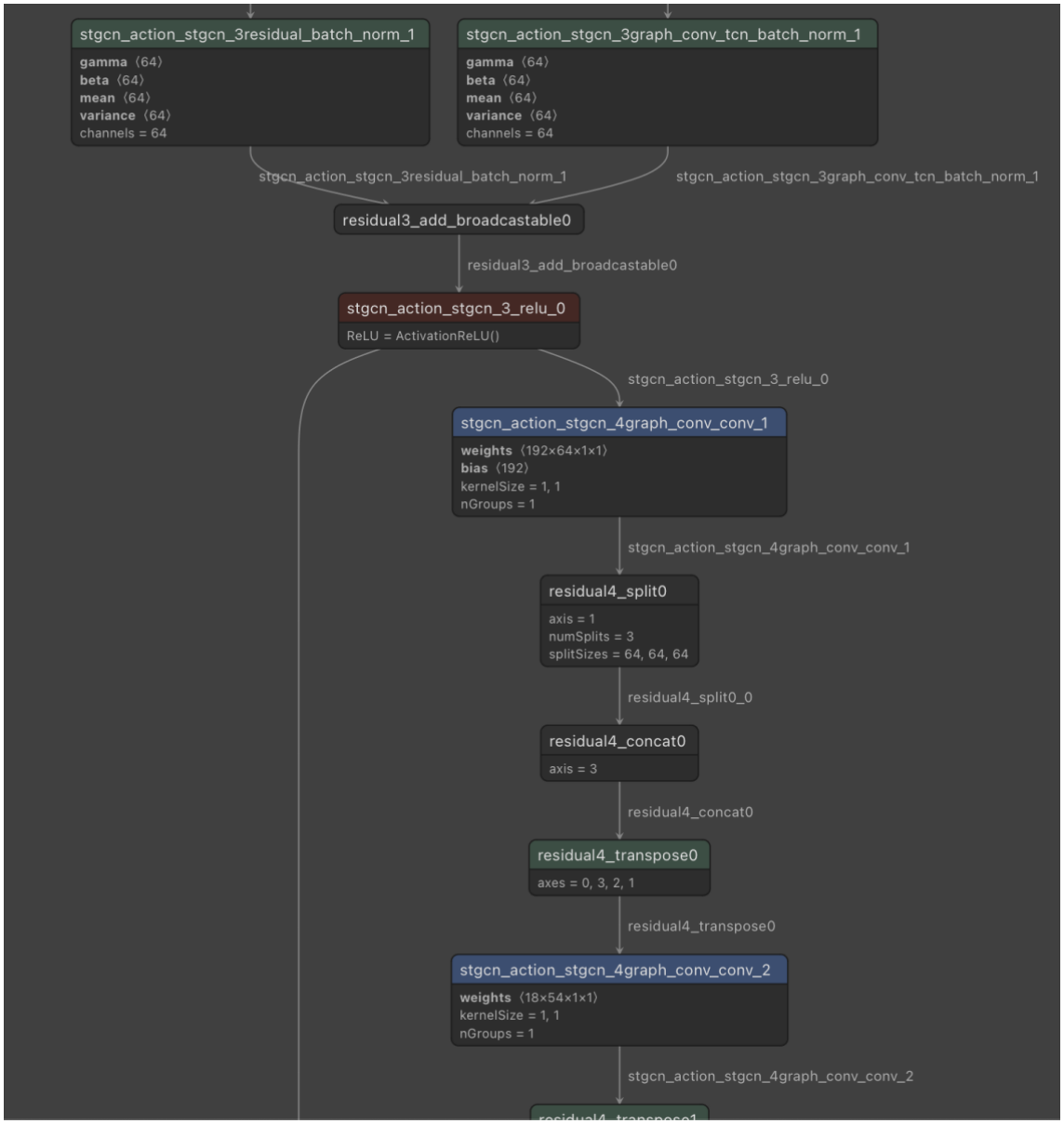

encode_pad
stgcn_action_stgcn_0graph_conv_tcn_conv_1
weights ⟨8×8×9×1⟩
bias ⟨8⟩
kernelSize = 9, 1
nGroups = 1
stgcn_action_stgcn_0graph_conv_tcn_conv_1
stgcn_action_stgcn_0graph_conv_tcn_batch_norm_1
gamma ⟨8⟩
beta ⟨8⟩
mean ⟨8⟩
variance ⟨8⟩
channels = 8
stgcn_action_stgcn_0graph_conv_tcn_batch_norm_1
stgcn_action_stgcn_0_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_0_relu_0
stgcn_action_stgcn_1graph_conv_conv_1
weights ⟨48×8×1×1⟩
bias ⟨48⟩
kernelSize = 1, 1
nGroups = 1
residual1_split0
axis = 1
numSplits = 3
splitSizes = 16, 16, 16
residual1_split0_0
residual1_concat0
axis = 3
residual1_concat0
residual1_transpose0
axes = 0, 3, 2, 1
residual1_transpose0
stgcn_action_stgcn_1graph_conv_conv_2
weights ⟨18×54×1×1⟩
stgcn_action_stgcn_0_relu_0

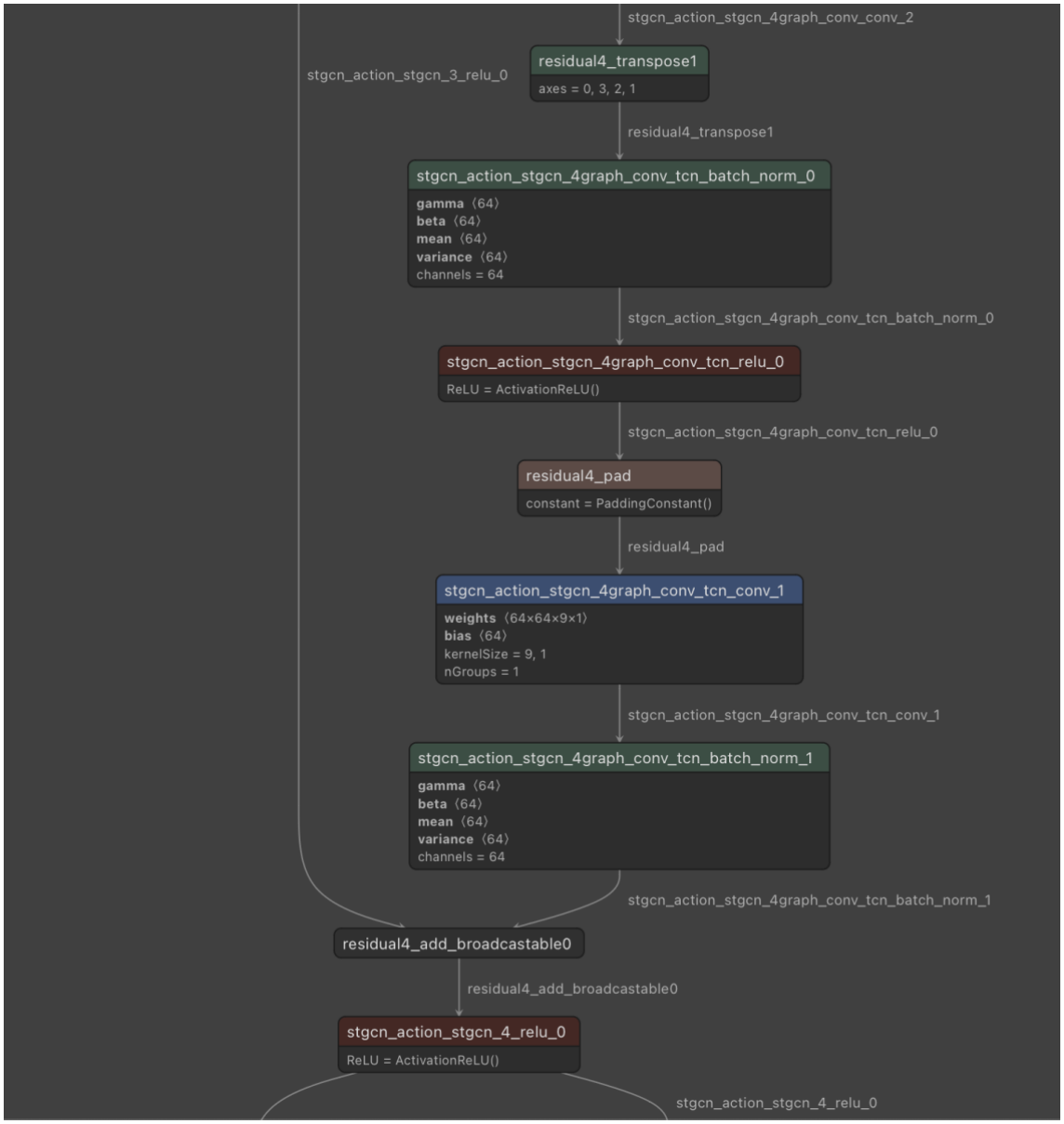

residual1_transpose0
stgcn_action_stgcn_0_relu_0
stgcn_action_stgcn_1graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_1graph_conv_conv_2
residual1_transpose1
axes = 0, 3, 2, 1
residual1_transpose1
stgcn_action_stgcn_1graph_conv_tcn_batch_norm_0
gamma ⟨16⟩
beta ⟨16⟩
mean ⟨16⟩
variance ⟨16⟩
channels = 16
stgcn_action_stgcn_1graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_1graph_conv_tcn_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_1graph_conv_tcn_relu_0
residual1_pad
constant = PaddingConstant()
residual1_pad
stgcn_action_stgcn_1residual_conv_1
weights ⟨16×8×1×1⟩
bias ⟨16⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_1graph_conv_tcn_conv_1
weights ⟨16×16×9×1⟩
bias ⟨16⟩
kernelSize = 9, 1
nGroups = 1
stgcn_action_stgcn_1residual_conv_1
stgcn_action_stgcn_1graph_conv_tcn_conv_1
stgcn_action_stgcn_1residual_batch_norm_1
gamma ⟨16⟩
beta ⟨16⟩
mean ⟨16⟩
variance ⟨16⟩
channels = 16
stgcn_action_stgcn_1graph_conv_tcn_batch_norm_1
gamma ⟨16⟩
beta ⟨16⟩
mean ⟨16⟩
variance ⟨16⟩
channels = 16
stgcn_action_stgcn_1residual_batch_norm_1
stgcn_action_stgcn_1graph_conv_tcn_batch_norm_1
residual1_add_broadcastable0

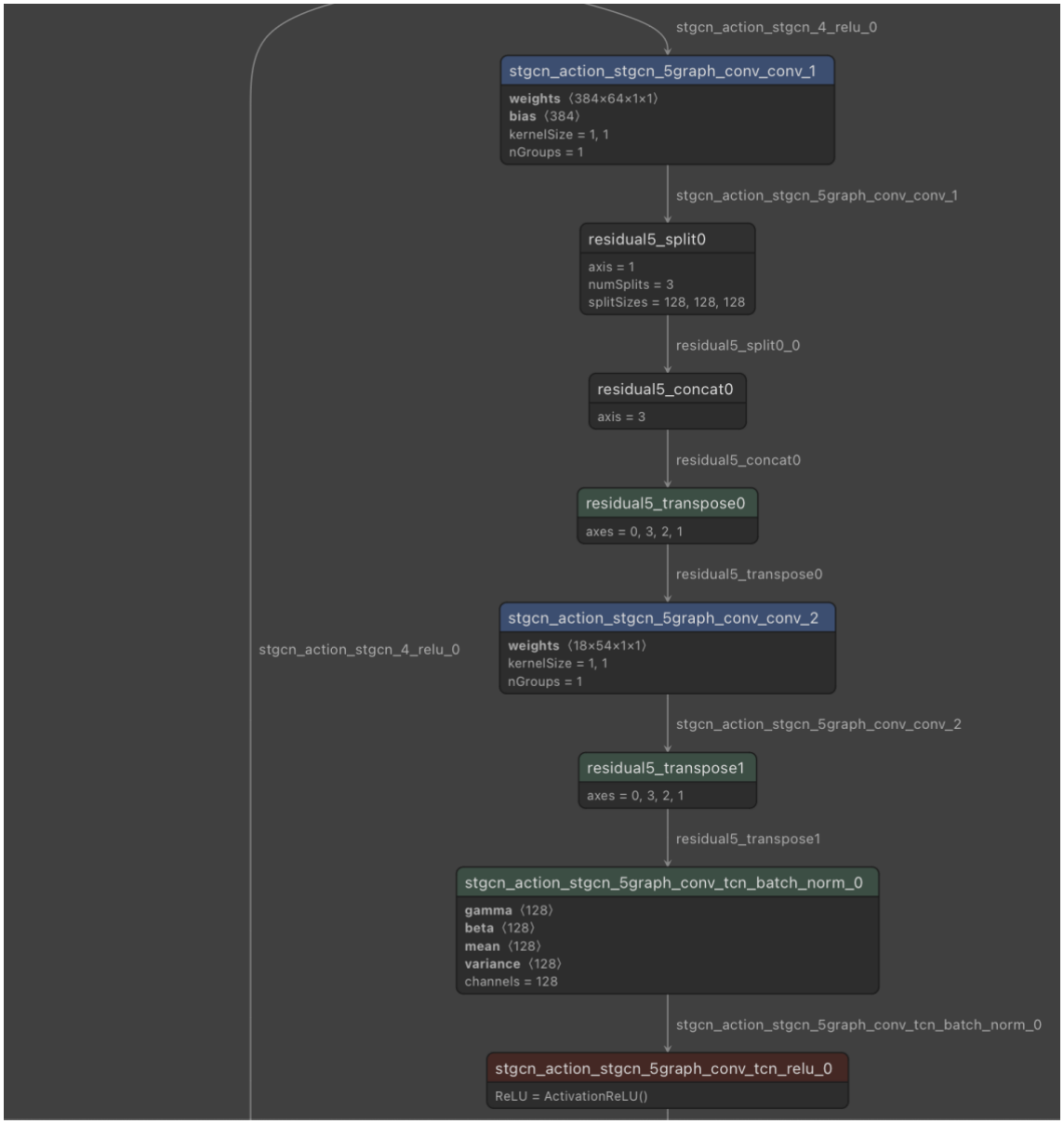

residual1_add_broadcastable0
stgcn_action_stgcn_1_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_1_relu_0
stgcn_action_stgcn_2graph_conv_conv_1
weights ⟨96×16×1×1⟩
bias ⟨96⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_2graph_conv_conv_1
residual2_split0
axis = 1
numSplits = 3
splitSizes = 32, 32, 32
residual2_split0_0
residual2_concat0
axis = 3
residual2_concat0
residual2_transpose0
axes = 0, 3, 2, 1
residual2_transpose0
stgcn_action_stgcn_1_relu_0
stgcn_action_stgcn_2graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_2graph_conv_conv_2
residual2_transpose1
axes = 0, 3, 2, 1
residual2_transpose1
stgcn_action_stgcn_2graph_conv_tcn_batch_norm_0
gamma ⟨32⟩
beta ⟨32⟩
mean ⟨32⟩
variance ⟨32⟩
channels = 32

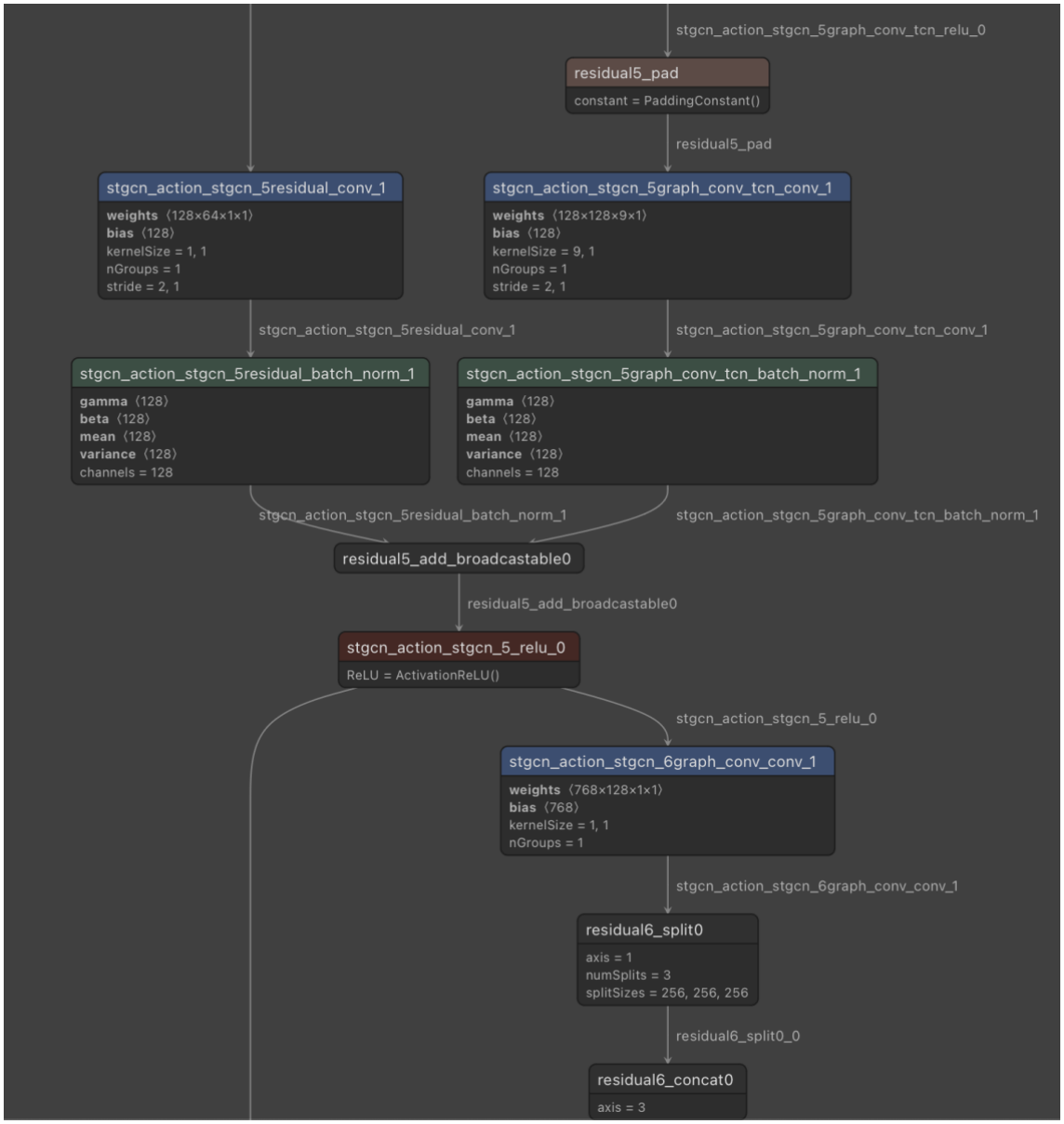

residual3_split0_0
residual3_concat0
axis = 3
residual3_concat0
residual3_transpose0
axes = 0, 3, 2, 1
residual3_transpose0
stgcn_action_stgcn_3graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_2_relu_0
stgcn_action_stgcn_3graph_conv_conv_2
residual3_transpose1
axes = 0, 3, 2, 1
residual3_transpose1
stgcn_action_stgcn_3graph_conv_tcn_batch_norm_0
gamma ⟨64⟩
beta ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
channels = 64
stgcn_action_stgcn_3graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_3graph_conv_tcn_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_3graph_conv_tcn_relu_0
residual3_pad
constant = PaddingConstant()
residual3_pad
stgcn_action_stgcn_3residual_conv_1
weights ⟨64×32×1×1⟩
bias ⟨64⟩
kernelSize = 1, 1
nGroups = 1
stride = 2, 1
stgcn_action_stgcn_3graph_conv_tcn_conv_1
weights ⟨64×64×9×1⟩
bias ⟨64⟩
kernelSize = 9, 1
nGroups = 1
stride = 2, 1
stgcn_action_stgcn_3residual_conv_1
stgcn_action_stgcn_3graph_conv_tcn_conv_1

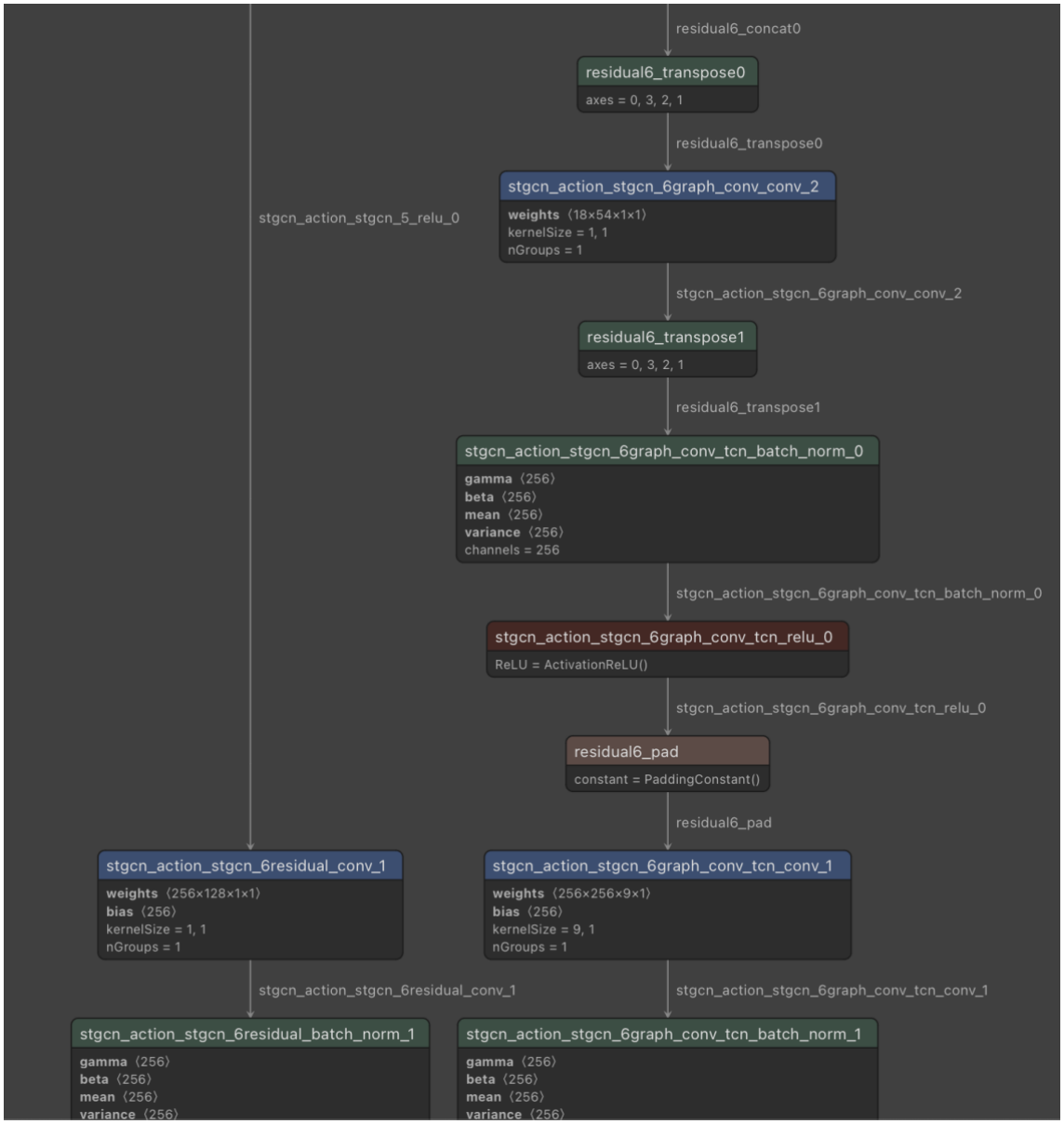

stgcn_action_stgcn_2graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_2graph_conv_tcn_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_2graph_conv_tcn_relu_0
residual2_pad
constant = PaddingConstant()
residual2_pad
stgcn_action_stgcn_2residual_conv_1
weights ⟨32×16×1×1⟩
bias ⟨32⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_2graph_conv_tcn_conv_1
weights ⟨32×32×9×1⟩
bias ⟨32⟩
kernelSize = 9, 1
nGroups = 1
stgcn_action_stgcn_2residual_conv_1
stgcn_action_stgcn_2graph_conv_tcn_conv_1
stgcn_action_stgcn_2residual_batch_norm_1
gamma ⟨32⟩
beta ⟨32⟩
mean ⟨32⟩
variance ⟨32⟩
channels = 32
stgcn_action_stgcn_2graph_conv_tcn_batch_norm_1
gamma ⟨32⟩
beta ⟨32⟩
mean ⟨32⟩
variance ⟨32⟩
channels = 32
stgcn_action_stgcn_2residual_batch_norm_1
stgcn_action_stgcn_2graph_conv_tcn_batch_norm_1
residual2_add_broadcastable0
residual2_add_broadcastable0
stgcn_action_stgcn_2_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_2_relu_0
stgcn_action_stgcn_3graph_conv_conv_1
weights ⟨192×32×1×1⟩
bias ⟨192⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_3graph_conv_conv_1
residual3_split0
axis = 1
numSplits = 3
splitSizes = 64, 64, 64

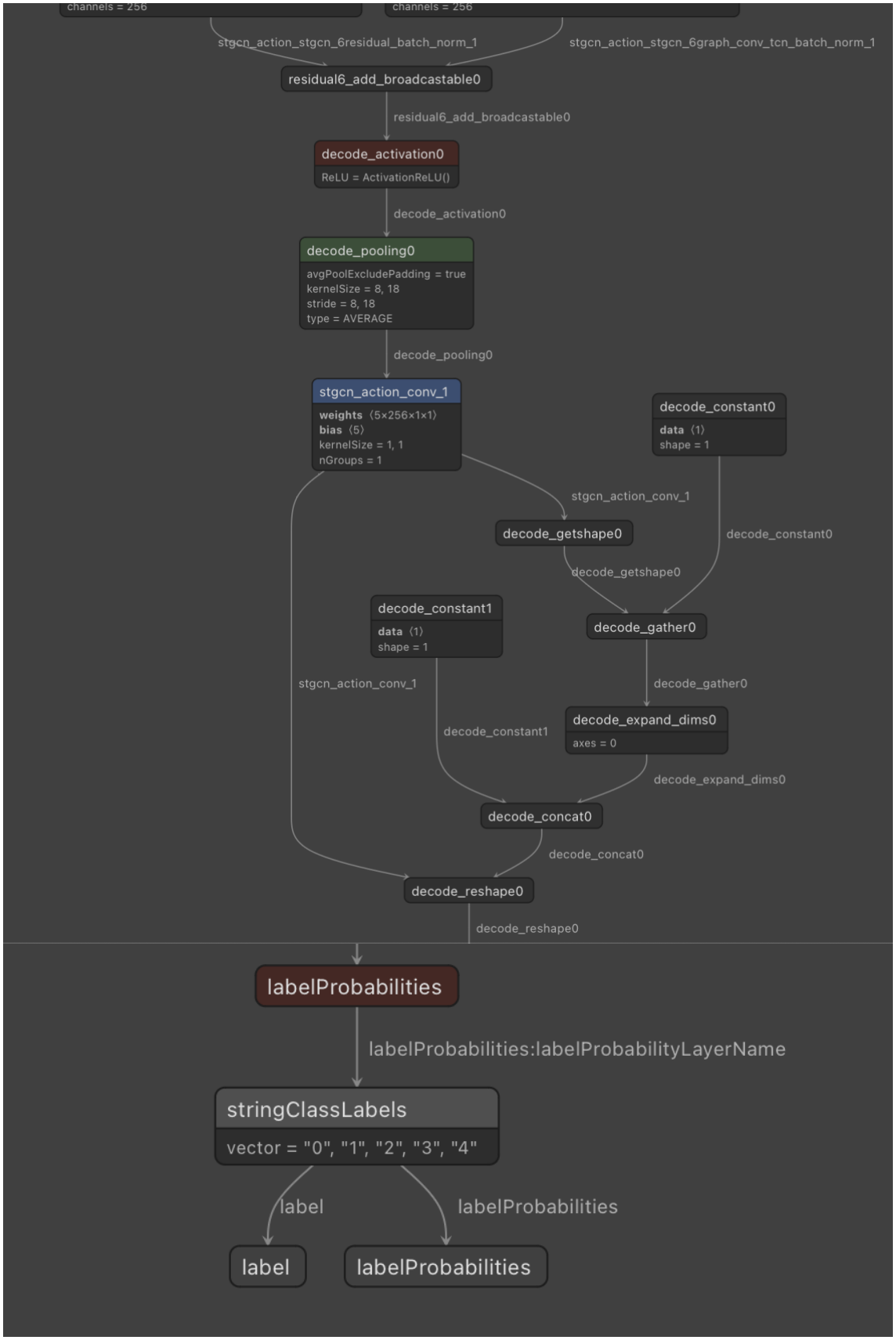

stgcn_action_stgcn_3residual_batch_norm_1
gamma ⟨64⟩
beta ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
channels = 64
stgcn_action_stgcn_3graph_conv_tcn_batch_norm_1
gamma ⟨64⟩
beta ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
channels = 64
stgcn_action_stgcn_3residual_batch_norm_1
stgcn_action_stgcn_3graph_conv_tcn_batch_norm_1
residual3_add_broadcastable0
residual3_add_broadcastable0
stgcn_action_stgcn_3_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_3_relu_0
stgcn_action_stgcn_4graph_conv_conv_1
weights ⟨192×64×1×1⟩
bias ⟨192⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_4graph_conv_conv_1
residual4_split0
axis = 1
numSplits = 3
splitSizes = 64, 64, 64
residual4_split0_0
residual4_concat0
axis = 3
residual4_concat0
residual4_transpose0
axes = 0, 3, 2, 1
residual4_transpose0
stgcn_action_stgcn_4graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_4graph_conv_conv_2


stgcn_action_stgcn_4graph_conv_conv_2
stgcn_action_stgcn_3_relu_0
residual4_transpose1
axes = 0, 3, 2, 1
residual4_transpose1
stgcn_action_stgcn_4graph_conv_tcn_batch_norm_0
gamma ⟨64⟩
beta ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
channels = 64
stgcn_action_stgcn_4graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_4graph_conv_tcn_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_4graph_conv_tcn_relu_0
residual4_pad
constant = PaddingConstant()
residual4_pad
stgcn_action_stgcn_4graph_conv_tcn_conv_1
weights ⟨64×64×9×1⟩
bias ⟨64⟩
kernelSize = 9, 1
nGroups = 1
stgcn_action_stgcn_4graph_conv_tcn_conv_1
stgcn_action_stgcn_4graph_conv_tcn_batch_norm_1
gamma ⟨64⟩
beta ⟨64⟩
mean ⟨64⟩
variance ⟨64⟩
channels = 64
stgcn_action_stgcn_4graph_conv_tcn_batch_norm_1
residual4_add_broadcastable0
residual4_add_broadcastable0
stgcn_action_stgcn_4_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_4_relu_0


stgcn_action_stgcn_4_relu_0
stgcn_action_stgcn_5graph_conv_conv_1
weights ⟨384×64×1×1⟩
bias ⟨384⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_5graph_conv_conv_1
residual5_split0
axis = 1
numSplits = 3
splitSizes = 128, 128, 128
residual5_split0_0
residual5_concat0
axis = 3
residual5_concat0
residual5_transpose0
axes = 0, 3, 2, 1
residual5_transpose0
stgcn_action_stgcn_4_relu_0
stgcn_action_stgcn_5graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_5graph_conv_conv_2
residual5_transpose1
axes = 0, 3, 2, 1
residual5_transpose1
stgcn_action_stgcn_5graph_conv_tcn_batch_norm_0
gamma ⟨128⟩
beta ⟨128⟩
mean ⟨128⟩
variance ⟨128⟩
channels = 128
stgcn_action_stgcn_5graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_5graph_conv_tcn_relu_0
ReLU = ActivationReLU()


stgcn_action_stgcn_5graph_conv_tcn_relu_0
residual5_pad
constant = PaddingConstant()
residual5_pad
stgcn_action_stgcn_5residual_conv_1
weights ⟨128×64×1×1⟩
bias ⟨128⟩
kernelSize = 1, 1
nGroups = 1
stride = 2, 1
stgcn_action_stgcn_5graph_conv_tcn_conv_1
weights ⟨128×128×9×1⟩
bias ⟨128⟩
kernelSize = 9, 1
nGroups = 1
stride = 2, 1
stgcn_action_stgcn_5residual_conv_1
stgcn_action_stgcn_5graph_conv_tcn_conv_1
stgcn_action_stgcn_5residual_batch_norm_1
gamma ⟨128⟩
beta ⟨128⟩
mean ⟨128⟩
variance ⟨128⟩
channels = 128
stgcn_action_stgcn_5graph_conv_tcn_batch_norm_1
gamma ⟨128⟩
beta ⟨128⟩
mean ⟨128⟩
variance ⟨128⟩
channels = 128
stgcn_action_stgcn_5residual_batch_norm_1
stgcn_action_stgcn_5graph_conv_tcn_batch_norm_1
residual5_add_broadcastable0
residual5_add_broadcastable0
stgcn_action_stgcn_5_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_5_relu_0
stgcn_action_stgcn_6graph_conv_conv_1
weights ⟨768×128×1×1⟩
bias ⟨768⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_6graph_conv_conv_1
residual6_split0
axis = 1
numSplits = 3
splitSizes = 256, 256, 256
residual6_split0_0
residual6_concat0
axis = 3


residual6_concat0
residual6_transpose0
axes = 0, 3, 2, 1
residual6_transpose0
stgcn_action_stgcn_6graph_conv_conv_2
weights ⟨18×54×1×1⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_5_relu_0
stgcn_action_stgcn_6graph_conv_conv_2
residual6_transpose1
axes = 0, 3, 2, 1
residual6_transpose1
stgcn_action_stgcn_6graph_conv_tcn_batch_norm_0
gamma ⟨256⟩
beta ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
channels = 256
stgcn_action_stgcn_6graph_conv_tcn_batch_norm_0
stgcn_action_stgcn_6graph_conv_tcn_relu_0
ReLU = ActivationReLU()
stgcn_action_stgcn_6graph_conv_tcn_relu_0
residual6_pad
constant = PaddingConstant()
residual6_pad
stgcn_action_stgcn_6residual_conv_1
weights ⟨256×128×1×1⟩
bias ⟨256⟩
kernelSize = 1, 1
nGroups = 1
stgcn_action_stgcn_6graph_conv_tcn_conv_1
weights ⟨256×256×9×1⟩
bias ⟨256⟩
kernelSize = 9, 1
nGroups = 1
stgcn_action_stgcn_6residual_conv_1
stgcn_action_stgcn_6graph_conv_tcn_conv_1
stgcn_action_stgcn_6residual_batch_norm_1
gamma ⟨256⟩
beta ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩
stgcn_action_stgcn_6graph_conv_tcn_batch_norm_1
gamma ⟨256⟩
beta ⟨256⟩
mean ⟨256⟩
variance ⟨256⟩


channels = 256
channels = 256
stgcn_action_stgcn_6residual_batch_norm_1
stgcn_action_stgcn_6graph_conv_tcn_batch_norm_1
residual6_add_broadcastable0
residual6_add_broadcastable0
decode_activation0
ReLU = ActivationReLU()
decode_activation0
decode_pooling0
avgPoolExcludePadding = true
kernelSize = 8, 18
stride = 8, 18
type = AVERAGE
decode_pooling0
stgcn_action_conv_1
weights ⟨5×256×1×1⟩
bias ⟨5⟩
kernelSize = 1, 1
nGroups = 1
decode_constant0
data ⟨1⟩
shape = 1
stgcn_action_conv_1
decode_getshape0
decode_constant0
decode_getshape0
decode_constant1
data ⟨1⟩
shape = 1
decode_gather0
stgcn_action_conv_1
decode_gather0
decode_expand_dims0
axes = 0
decode_constant1
decode_expand_dims0
decode_concat0
decode_concat0
decode_reshape0
decode_reshape0
labelProbabilities
labelProbabilities:labelProbabilityLayerName
stringClassLabels
vector = "0", "1", "2", "3", "4"
label
labelProbabilities
label
labelProbabilities