UNIVERSITY OF THESSALY
SCHOOL OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE AND BIOMEDICAL INFORMATICS

Image-based pain classification with machine learning techniques

Katranzopoulou Maria

THESIS

Supervisor
Delibasis Konstantinos
Associate Professor

Lamia 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

Κατηγοριοποίηση πόνου από εικόνες προσώπου με τεχνικές μηχανικής μάθησης

Κατρανζοπούλου Μαρία

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Δελήμπασης Κωνσταντίνος
Αναπληρωτής Καθηγητής

Λαμία 2023

Κατηγοριοποίηση πόνου από εικόνες προσώπου με τεχνικές μηχανικής μάθησης

Κατρανζοπούλου Μαρία

Τριμελής Επιτροπή:

1. Δελήμπασης Κωνσταντίνος, Αναπληρωτής Καθηγητής (επιβλέπων)
2. Ιακωβίδης Δημήτριος, Καθηγητής
3. Τασουλής Σωτήριος, Επίκουρος Καθηγητής

Acknowledgements

I would like to express my sincere gratitude to all those who have supported and encouraged me throughout the course of this project.

First and foremost, I would like to thank my thesis advisor, Dr.Konstantinos Delibasis, for his guidance and support throughout the research process. His continuous encouragement and expertise were invaluable for the completion of this project. I would also like to thank the University of Thessaly, for providing me with the resources needed to accomplish this project.

Finally, I would like to extend my heartfelt appreciation to my family and friends who have been a constant source of encouragement and motivation. Their unwavering support, understanding, and patience have sustained me through the ups and downs of this journey. I am deeply grateful for their love and presence in my life.

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις (1), που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί χωρίς να τα περικλείω σε εισαγωγικά και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.

2. Δέχομαι ότι η αυτολεξεί παράθεση χωρίς εισαγωγικά, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ...../...../20.....

Ο – Η Δηλ.

(Υπογραφή)

(1) ≪Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Abstract

Affective computing is a field that focuses on the application of computer technology to enhance the algorithmic understanding of human emotions. Pain intensity estimation is an important aspect of this field, where computer algorithms can be used to predict the intensity of pain experienced by a person. In recent years, there has been a growing need for automatic pain assessment in healthcare as traditional subjective self-reports are not considered valid ways for pain estimation. Subjective pain evaluation can be influenced by various factors such as the patient's emotional state and level of consciousness. Therefore, the reliability of facial expressions as an indicator of pain has been investigated through multiple studies, providing evidence of the universality of pain expression through nonverbal communication.

This thesis aims to develop a model to classify pain intensity levels as "no-pain", "weak", or "strong" by automatic facial image analysis with the objective of improving personalized healthcare, psychological research, and pain management. In this study, we have implemented various machine learning models including Convolutional Neural Networks (CNNs), Random Forests (RFs), Support Vector Machines (SVMs), and Decision Trees (DTs) to estimate the pain intensity. To optimize the performance of these models, we carefully selected or tuned their hyperparameters. Two types of features were also extracted and used to train the simple classifiers: geometrical features based on fiducial points and image-based features, including hand-crafted Histogram of Oriented Gradients (HOG) features and image features learnt by deep learning architectures, such as CNNs. In addition, the preprocessing steps for CNNs include the detection of faces, the alignment, the resizing, and the normalization. The study took into consideration the imbalance problem, where the number of samples for different pain levels is unequal. To handle this issue different methods were explored in order to address the imbalance in both data and algorithm levels as well as different splitting strategies. In this study, the UNBC-McMaster Shoulder Pain Expression Archive Database was used as the dataset for the analysis of pain intensity which contains video recordings of people experiencing pain in their shoulder.

The results showed that the pre-trained CNNs outperformed the simple classifiers in terms of precision, recall, and F1-macro score. The study also revealed the limitations of the dataset, including the small size and the lack of diverse facial expressions, leading to limitations in the generalizability of the models. In conclusion, this thesis contributes to the understanding of pain classification using facial expressions and provides insights for future directions of improvement, such as collecting more diverse datasets and exploring more advanced deep learning models.

Keywords : Pain Intensity, Facial Expressions, Support Vector Machines, Random Forest, Decision Trees, CNNs, Imbalance Problem

Περίληψη

Η πληροφορική των συναισθημάτων είναι ένας τομέας που επικεντρώνεται στην εφαρμογή της τεχνολογίας των υπολογιστών για την κατανόηση των ανθρώπινων συναισθημάτων. Η εκτίμηση της έντασης του πόνου είναι μια σημαντική πτυχή αυτού του πεδίου, όπου αλγόριθμοι μπορούν να χρησιμοποιηθούν για την πρόβλεψη της έντασης του πόνου που βιώνει ένα άτομο. Τα τελευταία χρόνια, υπάρχει αυξανόμενη ανάγκη για την αυτόματη εκτίμηση του πόνου στην υγειονομική περίθαλψη, καθώς οι παραδοσιακές υποκειμενικές αυτοαναφορές δεν θεωρούνται έγκυροι τρόποι αξιολόγησης του πόνου. Η υποκειμενική εκτίμηση του πόνου μπορεί να επηρεαστεί από διάφορους παράγοντες, όπως η συναισθηματική κατάσταση του ασθενούς και το επίπεδο συνείδησης. Επιπλέον, πολλές μελέτες έχουν διερευνήσει την αξιοπιστία των εκφράσεων του προσώπου ως δείκτη εκτίμησης του πόνου και παρέχουν πληροφορίες για μια καθολική έκφραση του πόνου.

Η παρούσα πτυχιακή εργασία αποσκοπεί στην δημιουργία ενός μοντέλου για την ταξινόμηση του πόνου ως "καθόλου πόνος", "ασθενής πόνος" ή "ισχυρός πόνος" μέσω της ανάλυσης της εικόνας του προσώπου, με στόχο τη βελτίωση της εξατομικευμένης υγειονομικής περίθαλψης, της έρευνας στον τομέα της ψυχολογίας και της άμεσης κλινικής διαχείρισης του πόνου. Σε αυτή τη μελέτη, εφαρμόσαμε διάφορα μοντέλα μηχανικής μάθησης, συμπεριλαμβανομένων των Συνελικτικών Νευρωνικών Δικτύων (Convolutional Neural Networks), των τυχαίων δασών (Random Forests), των μηχανών διανυσμάτων υποστήριξης (Support Vector Machines) και των δέντρων απόφασης (Decision Trees) για την εκτίμηση της έντασης του πόνου. Για να βελτιστοποιήσουμε την απόδοση αυτών των μοντέλων, προσαρμόσαμε προσεκτικά τις υπερπαραμέτρους τους. Εξήχθησαν επίσης δύο τύποι χαρακτηριστικών : γεωμετρικά χαρακτηριστικά με βάση σημεία αναφοράς στο πρόσωπο και χαρακτηριστικά με βάση την εικόνα, συμπεριλαμβανομένων των χαρακτηριστικών Histogram of Oriented Gradients (HOG) και των χαρακτηριστικών εικόνας που μαθαίνονται από αρχιτεκτονικές βαθιάς μάθησης, όπως τα CNN. Επιπλέον, τα βήματα προεπεξεργασίας για την είσοδο των εικόνων στα CNNs περιλαμβάνουν την ευθυγράμμιση των εικόνων, την ανίχνευση των προσώπων, την αλλαγή μεγέθους και την κανονικοποίηση των εικόνων. Η μελέτη έλαβε υπόψη το πρόβλημα της άνισης κατανομής των κλάσεων, όπου ο αριθμός των δειγμάτων για τα διαφορετικά επίπεδα πόνου είναι διαφορετικός, με την κατηγορία "καθόλου πόνος" να περιέχει την πλειοψηφία των εικόνων. Για την αντιμετώπιση αυτού του προβλήματος διερευνήθηκαν διάφορες μέθοδοι τόσο σε επίπεδο δεδομένων όσο και σε επίπεδο αλγορίθμων, καθώς και διαφορετικές μέθοδοι διαχωρισμού των δεδομένων. Στην παρούσα μελέτη χρησιμοποιήθηκε ως σύνολο δεδομένων για την ανάλυση της έντασης του πόνου η βάση δεδομένων UNBC-McMaster Shoulder Pain Expression Archive Database, η οποία περιέχει βίντεο ατόμων που βιώνουν πόνο στον ώμο τους.

Τα αποτελέσματα έδειξαν ότι τα προ-εκπαιδευμένα CNNs υπερείχαν έναντι των απλών ταξινομητών όσον αφορά την ακρίβεια (precision), την ανάκληση (recall) και τη μετρική F1-macro. Η μελέτη επισήμανε επίσης τους περιορισμούς του συνόλου δεδομένων, συμπεριλαμβανομένου του μικρού μεγέθους και της έλλειψης ποικίλων εκφράσεων προσώπου, που περιορίζουν τη γενικευσιμότητα των μοντέλων. Συμπερασματικά, η παρούσα εργασία συμβάλλει στην κατανόηση της ταξινόμησης του πόνου από τις εκφράσεις του προσώπου.

Λέξεις-κλειδιά : Ένταση πόνου, Εκφράσεις Προσώπου, Support Vector Machines, Random Forest, Decision Trees, CNNs, Πρόβλημα άνισης κατανομής των κλάσεων

Contents

List of Figures

List of Tables

1 Introduction

1.1 Facial Expression and Pain

Facial expression is defined as a nonverbal communication method that involves the movement of facial muscles. In fact, it is one of the most effective, natural, and immediate ways for humans to express their emotions and intentions [15]. A facial expression is an integral part of the human communication process, as the face can communicate emotion before people verbalize or even acknowledge their emotions.

Since Darwin published his work in 1872 [16], the study of facial expression analysis has been a popular topic of investigation among behavioral scientists. Darwin claimed that facial expressions have their origins in the evolution of the human species. For instance, raising the eyebrows may have aided our ancestors in responding to unexpected environmental occurrences by enlarging their sight area and allowing them to observe more. Even if facial expressions no longer play a vital function in human survival, we nonetheless use them when something out of the ordinary occurs in our surroundings [17].

Additional research examines if the affinity between particular facial expressions and particular categories of emotion exists cross-culturally and probably universally. Darwin [16] performed a survey by sending letters to individuals living in different regions of the world, asking them to report on the emotion expressions they noticed when interacting with the natives of these regions. Based on his findings, he determined that expressions were really universal. Many years later, Paul Ekman conducted an observational research to see whether individuals globally have similar facial expressions during emotion expression and found a certain degree of universality [18].

The face not only contains the majority of the sensory apparatus (e.g., ears, eyes, mouth, and nose), but it also provides numerous health-related signals [19]. Due to behavioral responses, certain medical conditions such as pain alter the expression or appearance of the face. These alterations are a potential source of data for clinicians to gain insights into the health of patients [19].

Pain is defined by the International Association for the Study of Pain (IASP) as "an unpleasant sensory and emotional experience associated with actual or potential tissue damage, or described in terms of such damage" [20]. In other words, although pain is a physical sensation, it is also affected by psychological, environmental, and emotional factors [19]. The nervous system detects any discomfort or damage to tissues and sends a signal up the spinal cord to the brain causing people to respond to the anomaly [21]. Pain can be expressed either verbally or physically.

Multiple studies have presented evidence that facial expressions encode pain experience [22], [23]. The first significant attempt to develop a scientific foundation for the study of pain expression was made by Darwin [16]. He observed certain mouth and eye movements in humans experiencing pain: "the mouth may be closely compressed, or more commonly the lips are retracted, with the teeth clenched or ground together . . . ". More recently, according to Leventhal and Sharp's research [24], the forehead, brow, and eyelid regions of the face underwent specific alterations as a result of the extreme pain experienced during labor. Boucher [25] revealed evidence of distinct facial expressions for fear, grief, and pain. Prkachin, Currie, and Craig [26] had untrained observers watch videotapes of participants' responses to electric shocks of various intensities. They discovered that spectators were able to distinguish between current intensities based only on the nonverbal responses of the participants. Moreover, it is reported that the magnitude of certain facial movements rises as pain severity increases [27].

Research by Payen et al. [28] indicates that facial expression can reliably determine pain in severely ill sedated patients. A study of pain evaluation in elderly patients with severe dementia [29] found that individuals with pain had a stronger behavioral response than those without pain and that facial expressions can accurately measure pain in people who cannot communicate verbally. Different procedural pain behaviors such grimacing, wincing, and closing the eyes were identified in a study of people suffering procedural pain [30] showing a significant association between procedural pain and behavioral reactions.

Humans have always been able to identify pain from facial expressions. In fact, artists of the classic era were able to capture the sense of suffering through facial representations [31]. For example, the sculpture "Laocoön and His Sons" (Figure 1.1) depicts the mythological death of Trojan priest Laocoön and his two sons by sea serpents, where the use of the face to represent agonizing pain is remarkable. There are several studies that demonstrate the ability of humans from an early age to identify and interpret pain.

The ability to judge another person's pain heavily depends on visual cues. Observers regularly utilize these to assess pain in both adults and children [32], [33]. Eyes were identified as the most significant cue, followed by brows, eyelids, lips, head, forehead, and other body parts [34]. Caregivers of severely cognitively disabled children [34] and adults [35] use facial expression as an indicator of pain. Furthermore, children who are too young to use verbal and numerical pain assessments can nonetheless assess their own level of suffering by looking at cartoon or photographed faces, where the shape of the eyes, moth, and nasolabial furrow are used as indicators of intensity. [36].

1.2 Facial Action Coding System (FACS)

Ekman and Friesen [37] introduced in 1978 the Facial Action Coding System (FACS), a technique that allows any observable facial expression to be divided into small individual movements of muscles of the face named as Action Units (AUs). FACS assesses facial expression frequency and intensity without assigning emotional significance [38]. Facial expressions are characterized by 44 AUs [39]. Each AU is identified with a number and represents the contraction or relaxation of a single muscle or, less frequently, an independently moving muscle strand or a set of muscles that move as a unit. For instance, when the frontalis (pars medialis) muscle contracts, the inner portion of the eyebrows rises, which is denoted by AU1 [40].

The 30 of the 44 facial AUs are anatomically associated with the contractions of certain facial muscles: 12 for the upper face and 18 for the lower face [41]. Figure 1.2 provides an illustration of upper-face AUs. AUs may occur individually or in combinations. In fact, each AU is made to be independent from the others. Consequently, a face expression appearance can also be divided into combinations of single AUs.

Figure 1.1: Laocoön and His Sons,[1]

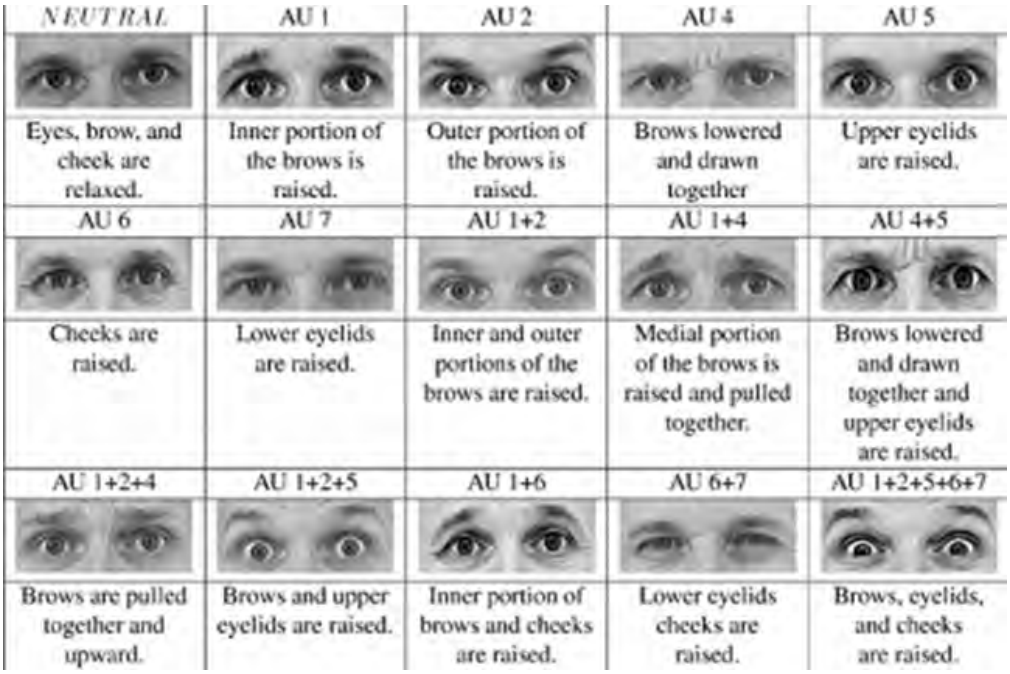


Figure 1.2: Upper face AUs and AUcombinations, where each action unit is demonstrated, along with the combinations of them, [2]

When AUs occur in combination, they can be additive, in which the combination does not alter the appearance of the individual AUs, or non-additive, in which the combination does affect the appearance of the components [2]. For example, the appearance of AU4 (brow lowering) differs depending on whether it occurs alone or in combination with AU1 (inner brow raising) [42]. When AU4 is present on its own, the brows become more contracted and lowered. However, during the combination of AU1 and AU4, the brows are drawn together and are raised.

There are also relationships between the intensities of several AUs. In certain AUs' co-occurrences, for instance, the dominant AU may totally cover the presence of a subordinate AU. Special criteria have been added to FACS so that the subordinate AU is not scored at all for certain combinations [43]. In addition to identifying which AUs are present on the face, the intensity of each AU must also be determined [44]. More precisely, the intensity is scored on a five-point ordinal scale, A-B-C-D-E, with A representing trace of the activity and E representing maximum evidence [45]. The primary AU codes, as well as their FACS names and muscle bases, are presented in Table 1.1.

1.3 FACS associated with Pain

There are many studies that investigate the presence of specific AUs during exposure to pain. LeResche [46] utilized FACS to characterize the facial expressions displayed in candid images of individuals experiencing severe pain. She reported that a typical pain expression included brow lowering, eyelid tightening, and horizontal lip stretching, with a deepening nasolabial furrow. Patrick et al. [47] examined the facial expressions of individuals reacting to the cold pressor test using the FACS. They observed significant increases in six facial actions associated with pain: cheek-raising/lid-tightening, upper lip rising, lip-corner tugging, lips parting, mouth opening and eye closure/blinking. Prkachin et al. [3] investigated reactions to pain caused by electric shock and concluded that, in addition to the previously mentioned facial actions, a considerable increase in brow lowering was also related with extreme pain.

Based on [3], AUs 6 (cheek-raising) and 7 (eyelid tightening) were combined into a single variable, orbit tightening. AUs 9 and 10 were similarly integrated into a single variable, levator tightening. Finally, AUs 25, 26, and 27 were merged into a four-point scale indicating the degree of mouth opening: 0 (no action), 1 (AU

Table 1.1: Description of several AUs together with the muscle name

AU	Description	Facial muscle
1	Inner Brow Raiser	Frontalis
2	Outer Brow Raiser	Frontalis
6	Cheek Raiser	Orbicularis oculi
7	Lid Tightener	Orbicularis oculi
9	Nose Wrinkler	Levator labii superioris
10	Upper Lip Raiser	Levator labii superioris
11	Nasolabial Deepener	Zygomaticus minor
12	Lip Corner Puller	Zygomaticus major
13	Cheek Puffer	Levator anguli oris
14	Dimpler	Buccinator
15	Lip Corner Depressor	Depressor anguli oris
16	Lower Lip Depressor	Depressor labii inferioris
17	Chin Raiser	Mentalis
20	Lip stretcher	Risorius w/ platysma

25, mouth open), 2 (AU 26, jaw drop), and 3 (AU 27, mouth stretch). These facial movements are observed during experimental and clinical pain [48]. However, this does not imply that a single facial expression of pain can be detected at all times and in all individuals [47].

The presence of the same actions associated with pain in the above experiments supported the concept of a characteristic and probably universal pain expression. Therefore, Prkachin [3] monitored facial reactions against four types of nociceptive stimulation, specifically cold, pressure, ischemia, and electric shock, using FACS. All four types of stimulation led to significant increases in four facial expressions: brow lowering (reflecting action of the corrugator muscle), orbit tightening(reflecting two actions of the orbicularis oculi), levator tightening(reflecting two actions of the levator labii superioris), and eye closure. This is also confirmed by a recent research involving shoulder-pain patients exposed to several unpleasant clinical range-of-motion tests [49]. AUs descriptors which correspond to the facial actions examined for their association with pain are shown in Table 1.2.

In Figure 1.3, a portrayal of pain expression in an adult face is depicted [3]. Figure 1.3(A) illustrates the tightening of the orbital muscles surrounding the eye, resulting to a constriction of the eye aperture and raising of the cheeks. Furthermore, in Figure 1.3(B) the corrugator and related muscle groups cause the eyebrows to lower and the bridge of the nose to wrinkle. Finally, the levator muscles raise the upper lip and can cause wrinkling at the side of the nose as it can be seen in Figure 1.3(C). Eventually, the eyelids may seen completely closed.

Table 1.2: AUs correlated to Pain Expression

AU	Facial Action
4	Brow Lowering
6	Cheek Raising
7	Eyelid Tighteninh
9	Nose Wrinkling
10	Upper Lip Raising
12	Oblique Lip Raising
20	Horizontal Lip Stretch
25	Lips Parting
26	Jaw Dropping
27	Mouth Stretching
43	Eye Closing

Figure 1.3: Potrayal of three phases in the development of pain expression, [3]

1.4 PSPI Pain Metric

In further research, Prkachin and Solomon (2008) [14] revalidated that the set of actions consisting of - brow lowering (AU4), orbital tightening (AU6 and AU7), levator contraction (AU9 and AU10), and eye closing (AU43) - transmitted the majority of information concerning pain. Expanding on this, Prkachin and Solomon proposed the Pain Intensity Metric (PSPI) metric.
The PSPI metric is defined as

$$Pain = AU4 + (AU6 \; or \; AU7) + (AU9 \; or \; AU10) + AU43. \tag{1.1}$$

With the exception of AU43, the PSPI scale of 16 points is derived from the sum of the AUs explaining pain on a six-point scale (0 = no activation, 5 = maximal activation). Due to its binary nature, AU43 is measured as 0 for eyes open and 1 for eyes closed. The calculation of pain will include the intensity of AU6 or AU7 based on which of the two has the higher intensity. This also applies to AU9 and AU10. Consequently, the maximum pain intensity will be 16 (5 + 5 + 5 + 1 = 16) [50].

PSPI metric is the only metric that can define pain on a frame-by-frame basis. It involves manual labeling of FACS by highly skilled observers. Manually classifying a single minute of video, can take hours, making it unsuitable for use in clinical settings when speed is of the essence.

1.5 Motivations

The main motive behind automatic pain intensity estimation is the need to quantify pain in an objective way. In clinical practice, pain is often assessed based on the patient's statement of its intensity, location, and duration. It is normally evaluated through self-report questionnaires or visual analog scales that require the patient's ability to communicate. Self-reporting is subjective, and it lacks detailed time information as whether pain is rising, decreasing, or spiking and can be greatly biased by a person's own prejudices and opinions [14]. In addition, some individuals may have trouble effectively communicating their pain levels, resulting in inconsistent or incorrect reports. Especially, people who are unable to express their pain experience, such as neonates, patients with cognitive impairments, and unconscious individuals, are disqualified for this procedure. Self-reports can also be affected by psychological factors such as depression, anxiety, and cognitive impairment, which makes it difficult to acquire an accurate pain intensity measurement. Furthermore, in care units, pain is usually monitored by nurses' judgments. Every few hours, nurses review the patients' responses and, if necessary, alter their pain medication. According to research undertaken in this field health practitioners tend to underestimate pain [51], [52]. Considering all the above, self-reports may not be the most accurate methods for pain assessment.

The use of automatic pain estimation can have many benefits in various fields such as healthcare, psychology, and robotics. In healthcare, automatic pain estimation has the potential to greatly improve personalized healthcare. The need for personalized pain management is rooted in the fact that individuals experience pain and respond to treatments differently as in the case of elderly people, children, or individuals with disabilities. Since pain is subjective it is essential to tailor pain management solutions to the specific needs of each patient and make more informed treatment decisions, ensuring that the patient receives the appropriate care in a timely manner. Automatic pain assessment can also play a vital role in the development of innovative pain management treatments. Researchers and healthcare practitioners can get significant insights into the effectiveness of various treatments by correctly assessing pain intensity, allowing them to make data-driven decisions about which techniques are most likely to succeed. As a result, the healthcare industry can move towards a more personalized and evidence-based approach to pain management

In the discipline of psychology, particularly in the subfields of clinical psychology and psychiatric research, automatic pain recognition can be useful in the study of pain-related emotions and behavior. For cognitive neuroscientists, automatic pain recognition could offer significant insight into the experiences of patients who struggle with expressing pain due to cognitive impairments such as in the case of stage V dementia patients or people with Broca's aphasia, who have lost their ability to communicate with language. Utilization of pain recognition could significantly improve therapeutic outcomes for these patients.

1.6 Aim and Objectives

The aim of this thesis is to develop a model that can accurately classify facial images as one of several pain intensity levels (e.g. no-pain, weak, strong). The main idea assumes that computer-assisted systems can continuously monitor patients and assist in identifying facial abnormalities that may be imperceptible to a human observer [53]. In this study, we want to examine the effectiveness of simple classifiers with hand-crafted features to solve the subjective multiclass problem as well as the effectiveness of Convolutional Neural Networks (CNNs) and especially pre-trained networks that have already worked on facial tasks. It is important to determine how informative the proposed features are in solving the problem of facial pain recognition.
The thesis's specific **objectives** are:

- to build a computer-vision based system for automated pain recognition and classification from facial images.
- To extract features from the facial images, such as geometric and appearance features, to use as inputs for machine learning models.
- To investigate the usefulness of simple classifiers, such as Support Vector Machines, Decision Trees, and Random Forests for pain intensity estimation, using the extracted features as inputs.
- To train several CNNs, including both custom-designed networks and pre-trained networks, fine-tuned on the pain intensity classification task.
- To compare the performance of traditional classifiers and deep learning approaches for pain estimation.

The specific **contributions** of this work can be summarized as follows :

- To understand the difficulties of the UNBC-McMaster shoulder pain expression archive database, such as its small sample size of patients and lack of representation of all pain intensities.
- To examine subject-wise and sequence-wise splitting strategies in order to address the imbalance problem.
- To examine different methods for solving the imbalance problem in the dataset, which is a common challenge in medical classification tasks.
- To experiment with different ways for hyper-parameter optimization, such as grid search, random search, and Bayesian optimization, in order to find the optimal hyperparameters for each classifier.

- To examine various applications of transfer learning including fine-tuning the entire network, freezing some layers, and feature extraction.
- To understand the limitations of the existing literature in the field and the challenges of pain classification using facial expressions.
- To investigate the most useful and practical definition of classes of pain to be identified by an automatic system.

2 Background

2.1 Traditional Classifiers

Traditional classifiers are fundamental tools in the field of machine learning. They have been regularly used to make predictions based on labelled data for decades. These classifiers include linear models such as Support Vector Machines, as well as nonlinear models such as Decision Trees and Random Forests. Despite the increasing use of deep learning models, traditional classifiers remain important tools in a variety of applications, including facial analysis tasks. In this study, we aim to evaluate the effectiveness of these models when applied to hand-crafted characteristics in order to recognize 3 pain intensity levels.

2.1.1 Support Vector Machines

Support Vector Machines (SVMs) were introduced by Cortes and Vapnik in 1995 [54]. SVMs determine an optimal hyperplane to separate the data into a discrete number of predefined classes [55]. SVM not only attempts to minimize an empirical cost function but also imposes an additional constraint on the location of the hyperplane. This condition states that the hyperplane needs to be positioned in such a way that the distances between classes are maximised in order to generalize [55]. SVM is based on statistical learning. Although mathematical algorithms are designed to classify data into two classes with a hyperplane as decision surface, SVMs can also be employed to classify nonlinear and multi-class data.

SVM model, in its basic form, is a linear binary classifier, which identifies a single boundary between two classes [55]. Consider a training set with N sample pairs: $\{\mathbf{x_i}, y_i\}, i = 1, ..., N$, where $y_i \epsilon \{-1, +1\}$ determining in which class each point $\mathbf{x_i}$ belongs to and $\mathbf{x_i} \epsilon R^p$.
In general, any hyperplane defined by a set of points $\mathbf{x}$ can be written as:

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{2.1}$$

where $\mathbf{w}$ is the normal vector perpendicular of the hyperplane.
Two parallel hyperplanes (one for each class) can separate a set of points into two classes without any points between them. These hyperplanes can be described by the equations:

$$\mathbf{w}^T\mathbf{x_i} + b = -1$$
$$\mathbf{w}^T\mathbf{x_i} + b = 1$$

These two hyperplanes are separated by a distance of $\frac{2}{\|\mathbf{w}\|^2}$. The goal of the algorithm is to maximise this distance. To exclude all points from the band, the following conditions must hold for all the observations:

$$\mathbf{w}\mathbf{x_i} - b \geq 1, \ \forall \ y_i = 1$$
$$\mathbf{w}\mathbf{x_i} - b \leq 1, \ \forall \ y_i = -1$$

Multiplying each equation by its corresponding y_i they are transformed into just one equation as following:

$$y_i(\mathbf{w}\mathbf{x_i} - b) \geq 1, \ \forall i, i = 1...N \tag{2.2}$$

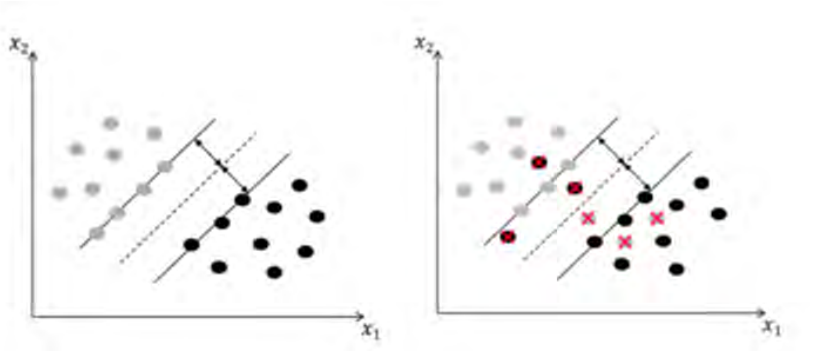


Figure 2.1: Hard (left) and soft (right) separating margins implemented on SVM, [4]

In addition, the optimization problem is solved:

$$\|\mathbf{w}\| \rightarrow min$$
$$y_i\left(\mathbf{w}\mathbf{x_i} - b\right) - 1 \geq 0)\ for\ 0 \leq i \leq n$$

In practice, data samples from different classes are not always linearly separable. Figure 2.1 presents two cases in which SVM constructs a separating hyperplane to successfully classify the majority of the instances when the data is perfectly separable (hard margin) and when it is not (soft margin). Thus, linear SVM needs some modifications to classify such data. To address the limitation of linear SVM, Cortes, and Vapnik proposed the soft margin and kernel trick methods [55].
In the soft margin case, a set of "soft" variables $\xi = \{\xi_1, \xi_2, ..., \xi_n\}$ is defined for the points that have been incorrectly categorized. Therefore, the optimation function is modified as :

$$\begin{aligned} \min_{\mathbf{w},\xi} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \right\} \\ \text{subject to } y_i(w\mathbf{x_i}) \geq 1 - \xi_i \end{aligned} \tag{2.3}$$

where C is the regularisation parameter that gives more weight to error minimization.

Furthermore, SVM includes kernel trick functionality that aids in mapping the original data into a higher-dimensional space prior to solving a specific task [55]. The selection of the kernel is very sensitive to the nature of the data. Some common kernel models are described in Table 2.1.

SVM could be extended to address multiclass classification by first dividing the single multiclass classification problem into multiple binary ones. The two most popular multiclass SVM methods are One-Versus-Rest (OVR) and One-Versus-One (OVO).

One-Versus-Rest SVM

The OVR approach builds one classifier for each class. In order to train each binary classifier, the examples from only one of the classes are considered positive, while the examples from all the other classes are considered negative. The multiclassifier output is enabled for the class whose binary classifier produces the highest output of all.

One-Versus-One SVM

OVO approach generates one binary classifier for each pair of distinct classes, resulting in a total of $\frac{M(M-1)}{2}$ binary classifiers for M classes. The binary classifier A_{ij} is trained with positive examples from class i and negative examples from class j. If classifier A_{ij} classifies a new example x as class i the vote for class i

Table 2.1: Kernel functions in Support Vector Machines

Kernel Function	Description	Mathematical Expression
Linear	The simplest kernel function, which just performs a dot product between the input vectors. It is used when the data is linearly separable.	$K(x, x^{'}) = x \cdot x^{'}$
Polynomial	A kernel function that creates a polynomial boundary between classes. It is used when the data is not linearly separable.	$K(x, x^{'}) = (gamma(x \cdot x^{'}) + r)^{d}$
Radial basis function (RBF)	A kernel function that creates a non-linear boundary between classes by transforming the input data into a higher dimensional space. It is used when the data is not linearly separable.	$K(x, x^{'}) = exp(-gamma \left\|x - x^{'}\right\|^{2})$
Sigmoid	A kernel function that creates a sigmoid boundary between classes. It is used when the data is not linearly seperable	$K(x, x') = tanh(gamma\ x \cdot x' + r)$

is increased by one. Otherwise, the class j vote is increased by one. The technique allocates the current example to the class with the most votes after each of the $\frac{M(M-1)}{2}$ binary classifiers casts its vote.

2.1.2 Decision Trees

In machine learning, Decision Trees (DTs) are used to classify a data instance into a predetermined set of classes based on its properties [5]. DTs were first introduced by Ross Quinlan in the 1970s [56]. Their simple implementation and their self-explanatory nature, allow them to be represented graphically as hierarchical tree-like structures, as illustrated in Figure 2.2 [5].

A DT is a classifier represented by a recursive partition on the instance space, with three distinct types of nodes: root node, internal nodes, and terminal nodes [5]. The root node is the starting point with no incoming edges. Each internal node represents a different value control condition for an attribute of the instances. An internal node divides the instance space into two or more partitions in order to reach the terminal nodes, which represent the optimal outcomes [5]. When building a DT, deciding under what circumstances to split the node is important. The two most common splitting criteria are Entropy and Gini.

Entropy measures the impurity or randomness in a bunch of observations and is used to evaluate the quality of a split. Entropy can be calculated by:

$$E = -\sum_{i=1}^{N} p_i log_2 p_i \tag{2.4}$$

,where p_i is the probability of selecting an instance at random from class i and N is the number of classes.

In a set with low entropy, most of its elements belong to the same class. In contrast a set with high entropy, includes elements that are uniformly distributed across several classes. The purpose of DT algorithms is to split the data in such a way that the entropy of the classes in the leaves is reduced.

The Gini is a statistical measure that determines the average impurity within a split, with the results being weighted according to the proportional size of the left and right branches. It calculates how likely a random element would be incorrectly classified if the sample is randomly labeled based on the given class

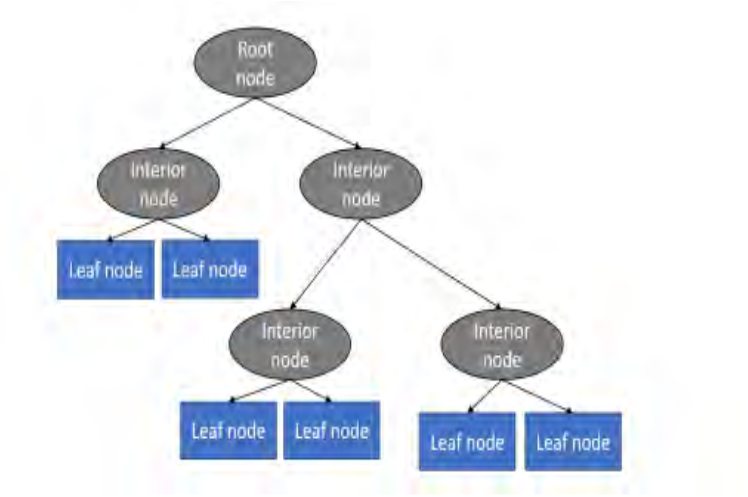


Figure 2.2: Decision Tree, [5]

distribution. The Gini can be defined as:

$$Gini = 1 - \sum p_i^2 \tag{2.5}$$

, where p_i is the proportion of items labeled with class i in the set.

DTs' framework can simply be described as the navigation of each instance from the tree's root down until it reaches any specific terminal node based on the results of the internal nodes [5].

2.1.3 Random Forest

Random Forest (RF) is an ensemble learning method that is used for both classification and regression problems. It was first introduced by Leo Breiman and Adele Cutler in 2001 [57]. Many decision trees, also known as weak learners, are created to solve the same problem [55].

RF uses a different random subset of data to construct each independent tree in order to reduce the correlation between the trees [55]. As a result, RF presents high robustness to noisy and unbalanced data. The algorithm creates a forest of Decision Trees, with each tree randomly selecting a subset of features at each split node instead of using all features. The Decision Trees' parameters are optimised during the training process, while the algorithm attempts to discover the optimal split nodes by comparing different feature values and determining which split results in the most homogeneous sub-nodes. This procedure is repeated for each split node, with the purpose of constructing a series of DTs that can accurately classify outcomes. Once all of the DTs have been trained individually, the algorithm takes the majority vote of their predictions and selects the class label with the highest number of votes. The implementation algorithm for the RF's method, is visualised in Figure 2.3.

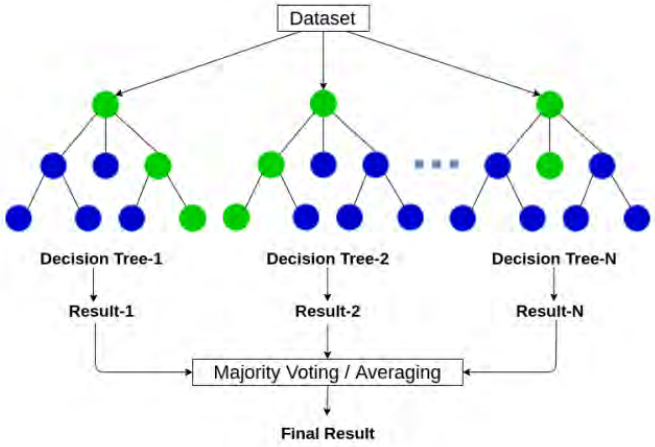


Figure 2.3: Diagrammatic representation of a random forest, [6]

2.2 Deep Learning Approaches

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have revolutionized the field of image and video analysis with their ability to effectively extract features from an image and make predictions. Their architecture, inspired by the human visual system, is composed of multiple layers, each with a specific role in processing the image. These networks have shown to achieve state-of-the-art performance in various image recognition tasks such as object detection, image classification and facial analysis. Furthermore, pre-trained CNNs can be fine-tuned on new tasks with minimal labeled data, making them a highly versatile tool for a wide range of real-world applications. In this thesis, we aim to delve into the potential of CNNs in pain intensity estimation. For this purpose, we conducted experiments using both simple CNN architectures and pre-trained models. A simple CNN architecture is depicted in Figure 2.4.

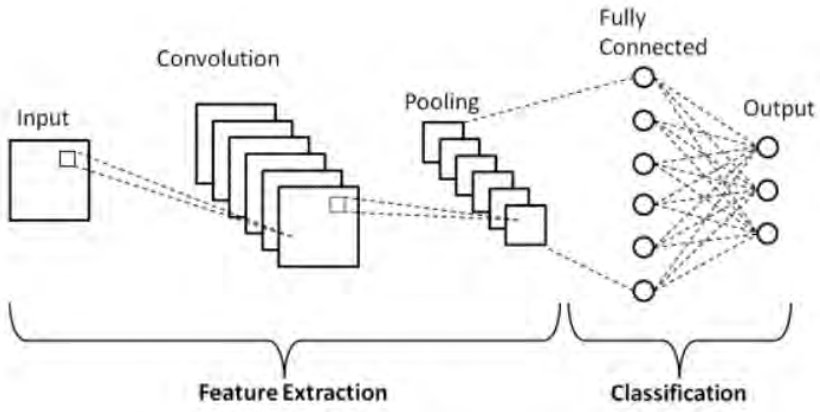


Figure 2.4: Architecture of basic Convolutional Neural Network, [7]

The main components of a CNN are:

1. *Convolutional layers*: These layers are employed in order to extract features from the input image. They consist of multiple learnable filters that are convoled with the input data to generate a set of feature maps. The filters are commonly 3×3 or 5×5 pixels in size with a 3rd dimension equal to the number of channels of the input, and they move across the image to implement the process of convolution.

The resulting feature maps are combined together for the final output. Padding and strides are two essential notions in convolutional layers. Stride is the step (in pixels) between 2 consecutive kernel positions across the input matrix. Padding defines the motion of the filter at the borders of the input. There are two types of padding: *valid padding*, which discards the input matrix's boundary pixels, and *zero padding*, which adds zeros to the borders such that the filter's output matches the input matrix.

2. *Pooling layers:* These layers are used to reduce the dimentionality of the feature maps [58]. This is advantageous since it reduces the amount of computational resources necessary to process the data while extracting the key characteristics from feature maps [58]. Pooling layers are placed after convolutional layers and are classified into two types: maximal pooling and average pooling layers. The most common pooling operation is max pooling, which selects the maximum value from a particular region of the feature map. It should be noted that none of the pooling layers have learnable parameters.

3. *Fully connected (FC) layers:* The output feature maps of the last convolution or pooling layer are often flattened (converted to a one-dimensional -1D- vector of numbers) and fed into a set of fully connected layers, in which each neuron of each FC layer is connected to all neurons of the next FC layer, by a learnable weight [59]. The number of output neurons in the last fully connected layer typically equals the number of classes [59].

4. *Dropout:* Dropout is a regularisation technique used to prevent overfitting. During training, the dropout layer removes randomly a fraction of units and their connections, which contributes to decreasing neuron co-adaptation. By lowering the number of parameters, the dropout layer enables the network to learn more robust and diversified characteristics.

5. *Batch normalization:* Batch normalisation is another way to reduce overfitting. During training, it normalises the input layer by modifying and scaling the activations [58]. It is often used following the convolutional layer and before the activation function.

There are many hyper-parameters that can be adjusted in order to improve the performance of the models. Some of the most important ones are:

1. *Learning Rate:* The learning rate specifies the step size at which the optimizer changes the weights of a neural network during training. A low learning rate can increase the execution time, whereas a high learning rate can cause the algorithm to miss the global optimum [21].

2. *Batch Size:* A batch is a subset of the training examples that are utilised to train the model during one iteration [21]. The majority of the time, the entire dataset cannot be fed into the neural network simultaneously, hence it must be separated into batches. The batch size specifies the number of training samples contained in a single batch.

3. *Number of Epochs:* An epoch represents a single iteration of the learning model on the entire dataset [21]. Epochs are particularly important in the iterative learning process. As long as the validation error is decreasing, the number of epochs can be raised. If the validation error does not improve after a certain number of epochs, the number of epochs should be decreased. A typical method for determining the ideal number of epochs is to employ approaches such as early stopping, in which training is terminated when performance on a validation dataset begins to deteriorate. Another method is to utilise a learning curve, which depicts a model's performance on a training and validation dataset as a function of the number of epochs. By evaluating the learning curve, it is possible to determine the point at which the model begins to overfit and select an appropriate number of epochs.

4. *Number of Hidden Units:* The number of hidden units determines the number of neurons in a neural network's hidden layer. A deep learning model's ability to learn and generalise depends on the number of hidden units in a layer.

5. *Number of Layers:* The number of layers specifies the depth of a neural network, or how many layers of neurons it has. A deeper network can learn more complex representations of the data, but it also increases the risk of overfitting.

2.2.2 Simple CNN

Many CNNs have been constructed to address the issue of automatic pain assessment. The architecture shown in Table 2.2, was proposed by Semwal et al.[13] and implemented to meet our project's requirements. The proposed architecture utilizes 4 convolutional layers.

In this architecture, images are initially preprocessed and resized to 96 × 96 × 3 pixels. After that, they are fed into the CNN model, which processes them as follows:

1. Images go through two convolutional layers, with 32 3×3 filters, a stride of 1 and a padding of 1, followed by a max pooling layer with a filter size of 2×2. A dropout layer is placed right after the first convolutional layer. The value for dropout is set to 0.2.
2. The resulting tensors go through an additional convolutional layer, with 16 3×3 filters, a stride of 1 and a padding of 1, followed by a max pooling layer with filter size of 2×2.
3. The resulting tensors go through the last convolutional layer, with 16 3×3 filters, a stride of 1 and a padding of 1, followed by a max pooling layer with a filter size of 2×2.
4. The resulting feature maps are flattened and converted to a fully connected layer of 16×12×12 = 2304 neurons, which is connected to a second fully connected layer of 512 neurons, which is finally connected to the output layer of three neurons, corresponding to the three classes. A dropout layer is placed before the output layer. The value for dropout is set to 0.5.

At each convolutional step, the ReLU activation function has been employed.

Layer Number	Layer	Filter size/values	Output Shape
L_1	Input		96×96×3
L_2	$Conv_1$	3×3	96×96 ×32
L_3	$Dropout_1$	0.2	96×96×32
L_4	$Conv_2$	3×3	96×96×32
L_5	$Maxpool_1$	2×2	48×48×32
L_6	$Conv_3$	3×3	48×48×16
L_7	$Maxpool_2$	2×2	24×24×16
L_8	$Conv_4$	3×3	24×24×16
L_9	$Maxpool_3$	2×2	12×12×16
L_{10}	$Flatten_1$	-	2304
L_{11}	$Dense_1$	-	512
L_{12}	$Dropout_2$	0.5	-
L_{13}	$Dense_2$		None, 3

Table 2.2: CNN architecture proposed by [13]

2.2.3 Deep Residual Neural Network

Due to their numerous layers, deep neural networks tend to overfit. Simply stacking layers together in order to increase the network's depth may result in the well-known vanishing gradient problem. In the process of the gradient being back-propagated to earlier layers, repeated multiplication may result in an extremely small gradient, leading to performance degradation [60]. The Deep Residual Learning Framework introduced by He et al. [8] is used to tackle the degradation problem. Deep Residual Neural Networks (ResNets) are made up of sequentially stacked residual blocks, visualized in 2.5.

Typically, a stack of convolutional layers receives the input x and processes it. However, in addition to this, a shortcut is implemented. Shortcut connections are interlayer connections that skip one or more layers. This shortcut connection is an identity function, where its output equals its input [61]. Rather than learning a direct mapping of

$$x \to H(x), \tag{2.6}$$

the network learns a residual mapping as:

$$\mathcal{F}(x) = H(x) - x \tag{2.7}$$

where, $\mathcal{F}(x)$ represents the stacked non-linear layers, and x represents the identity function (input = output)[60]. According to the author [8], optimising the residual mapping function $\mathcal{F}(x)$ is relatively easier than optimizing the original, mapping $H(x)$ [60].

In this thesis, two different variants of ResNet models were used, namely ResNet-18 and ResNet-50 that have 18 and 50 convolutional layers respectively. Both networks accept images of dimension 224 × 224. The ResNet architectures begin with two layers: a 7×7 convolution layer and a 3×3 max-pooling layer with a stride of 2. The residual blocks in ResNet-50 utilize a combination of 1×1, 3×3, and 1×1 convolutional layers, while the residual blocks in ResNet-18 utilize a combination of two 3×3 convolutional layers. A detailed comparison of the two architectures is given in Table 2.3.

Table 2.3: ResNet-18 and ResNet-50 architectures, source [8]

layer name	output size	18-layer	50-layer
*conv*1	112 × 112	7 × 7, 64, stride = 2	
*layer*1	56 × 56	3 × 3 max pool, stride 2	
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}$ x 2	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}$ x 3
*layer*2	28 × 28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}$ x 2	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}$ x 4
*layer*3	14 × 14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}$ x 2	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}$ x 6
*layer*4	7 × 7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}$ x 2	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}$ x 3
	1 × 1	average pool, 1000-d fc, softmax	

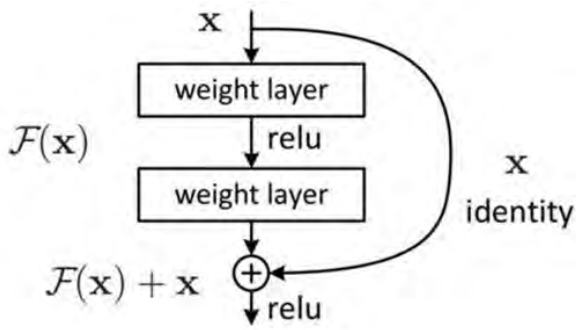


Figure 2.5: Residual block ,[8]

Pre-trained ResNe-50

Most research in the field of face recognition focuses on individual tasks without considering how to acquire a universal face representation that could be useful for multiple facial analysis tasks. However, the findings of Bulat et al. [62] indicate that using unsupervised pre-training to gain facial representations can improve performance on related facial tasks [63]. This means that it is not always necessary to collect a large and curated dataset when exploring a new human face analysis subtopic [64]. Instead, the robust pre-trained neural networks can be beneficial for future vision applications [64].

According to [62], they investigated large-scale unsupervised learning on facial images in order to learn a universal, task-independent facial representation. To achieve this, they used the SwAV algorithm, which clusters the data while enforcing uniformity between the cluster assignments generated for different augmentations (or "views") of the same image [62]. The method could be seen as a way to compare different views of an image by comparing their cluster assignments rather than their features.

As part of our work, we chose to experiment with the pre-trained network provided by Bulat. The network is based on the ResNet-50 and trained in a self-supervised manner on the full VGG-Face Dataset consisting of 2,622 identities [65].

Pre-trained ResNet-18

Pikoulis et al. worked on video-based emotion recognition by leveraging visual context in the form of scene characteristics and attributes [66]. The proposed method is based on late score fusion between a modified version of a Temporal Segment Network (TSN) and a pre-trained Spatial-Temporal Graph Convolutional Network ST-GCN [66]. One of the most significant contributions of this paper is the inclusion of multiple input streams that effectively encode bodily, contextual, facial, and generic scene-related features, enhancing the model's ability to understand the surrounding environment and emotion in general. A ResNet-18 model, pre-trained on the AffectNet database is determined as the facial feature extractor. AffectNet is the world's largest facial expression database, with over 1 million images annotated for the presence of eight different facial expressions, including neutral, happy, angry, sad, fear, surprise, disgust, and contempt along with the severity of valence and arousal. The pre-trained ResNet-18 was adapted appropriately to address our problem.

2.2.4 Transfer Learning in Deep Learning

Transfer learning is the process of improving learning for a new task by transferring knowledge from a previously learned, related task [67], [68]. More specifically, a model trained on one dataset is used to initialize a model trained on another different dataset [69].
There are two approaches for transfer learning with a pre-trained network:

1. **Fine-tuning:** Fine-tuning involves using a pre-trained model as a starting point and then updating the model's weights. CNNs frequently end up learning low-level features such as edges, colors, shapes which are shared by various types of images. Therefore, rather than starting from scratch, it is frequently preferable to use the weights from another network trained to perform a similar task [70], [71]. In cases where the target task differs from the source task, the output layer of the pre-trained model needs to be replaced with a new one tailored to the target task. All of the layers can be re-trained or some of the initial layers can be left as-is. However, the number of layers that could be re-trained (unfreeze) depends on the similarity of the tasks and the amount of data available for the new task. Technically, we must freeze more layers when we have a small dataset.

2. **Feature extraction:** Feature extraction involves using a pre-trained model as a feature extractor. This entails removing the last layer and using the remaining layers' output as input to a new classifier, which is trained from scratch on the target task. During training, the weights of the convolutional base are frozen, and only the weights of the new classifier are updated.

Additionally, deep neural network training is a time-consuming task due to the high computational demands. In such cases, employing transfer learning techniques could provide significant benefits in terms of both computational complexity and the need for a large dataset [68].

3 Related Work

The recognition of pain through the analysis of facial features is a complex task that has been the focus of significant research in the field of healthcare, with various approaches and methods proposed in the literature. This method may provide a non-invasive and objective way to measure pain that can be helpful in a clinical setting when self-report may be unreliable or difficult to acquire. The research on this topic has revealed that facial expressions can provide useful information about the intensity, duration, and different types of pain. However, despite the progress made in this area, further research is still needed to improve the accuracy and reliability of pain recognition methods, particularly in a clinical setting.

Binary Classification

Several approaches have been developed for pain recognition as a problem of binary classification, with the objective of distinguishing between pain and non-pain expressions.

For instance, Ashraf et al. [72] used the Active Appearance Models (AAMs) to model the variations in the shape and texture of a face. They extracted three types of features for each image: the 68 points of the similarity normalized shape representation (S-PTS), the similarity normalized appearance representation to mask all images based on the base shape (S-APP) and the canonical appearance representation to remove the shape variation with respect to the base shape (C-APP). They experiment with SVMs to solve the binary classification task. Similarly using the AAMs, Lucey et al. [73] extracted the similarity normalized shape, (S-PTS), the canonical normalized appearance (S-APS) and proceed also to their fusion. They performed two sets of experiments comparing direct versus indirect pain detection for each frame. In the first experiment, they created a direct pain detection system that determined whether a frame contained pain. In the second experiment, they developed an indirect pain detection system in order to detect AUs and then combined the outputs of each AU through logistical linear regression (LLR) fusion. In both experiments, linear SVM classifiers were trained.

Khan et al. [74] extracted shape information using Pyramid Histograms Of Orientation Gradients (PHOG) and appearance features using Pyramid Local Binary Pattern (PLBP). SVM, Decision Tree (DT), Random Forest (RF), and 2 nearest neighbors (2NN), have been used to test the performance of this method. Pedersen [75] proposed a discriminative feature extractor for automatic pain detection that is similar to an auto-encoder and is trained with a combined loss function that balances the reconstruction and classification errors. The resulting features are then fed to a linear SVM for binary classification. Furthermore, The method proposed by Yang et al [76] involves the extraction of 3 spatiotemporal local descriptors (STLDs): Local Binary Patterns (LBP-TOP), Local Phase Quantization (LPQ-TOP) and Binarized Statistical Image Features (BSIF-TOP) from facial videos using Three Orthogonal Planes (TOP). The resulting feature vectors are fed individually or combined using early fusion to support vector machines (SVM), in order to classify the video frames or the entire sequences as either painful or non-painful.

The study in [77] aims to compare the performance of ELM and SVM giving as input the AAM landmark points. The experiments were conducted in both the frame and sequence levels. Additionaly, Chen et al. [78] proposed a pain detection framework for videos based on spatial and dynamic features at various time scales (frame, segment, and sequence). At first, the authors utilized Histogram of Oriented Gradients (HOG) of fiducial points (P-HOG) and trained an SVM to detect the presence or absence of pain at the frame level. Secondly, at the segment level, an SVM is trained with HOG from Three Orthogonal Planes (HOG-TOP). In addition, a max pooling strategy to get the global P-HOG and HOG-TOP and a multiple

kernel fusion to combine the two types of global features are applied to represent the entire video sequence. Then, an SVM with multiple kernels is trained to detect pain events at the sequence level. Finally, in order to locate pain events in a video, an efficient probabilistic fusion method is proposed to combine the results of the three previous distinct tasks.

Multiclass Classification

Other works are not limited to binary classification but are primarily concerned with pain intensity assessment. These works frequently employ the Pain Intensity metric developed by Prkachin and Solomon (PSPI).

Hammal et al. [79] devised a four-level pain intensity classification approach. First, active appearance modeling was used to extract the canonical normalised appearance of the face (CAPP) which is then passed through a set of Log-Normal filters. Four distinct SVM models were trained for each class on the frame level. In [80] shape features such as locations of characteristic facial point features, appearance features including Discrete Cosine Transform (DCT) and Local Binary Patterns (LBP), and their late fusion were used to train Relevance Vector Regression (RVR) models for estimating continuous pain intensity. Furthermore, based on the Conditional Ordinal Random Field (CORF) model, Rudovic et al. [81] designed a model for estimating 6 pain intensities by incorporating heteroscedasticity. This is accomplished by allowing the variance in the CORF's ordinal probit model to change depending on the input features. The model is able to account for individual differences in pain expressiveness. Local Binary Patterns(LBPs) are used as features.

Using geometric features derived from 22 facial characteristic points, Zafar and Khan presented a multi-stage classification scheme [82]. Action units and their intensities are identified in the first stage using K-NN. In the second stage, the predicted AU intensities are combined, determining pain intensity on a 16-point scale. Moreover, in the feature extraction process, Irani et al. [83] employed steering filters that are separable to spatial and temporal domains and measured the energies released by the facial muscles during pain to classify the pain into 3 categories. Rathee et al. [84] proposed the use of Thin Plate Spline (TPS) mapping for modeling the deformation of facial features and the Distance Metric Learning (DML) method for measuring the distance between features corresponding to different intensities of pain. The mapped data is fed to SVM for pain detection on a 16-point scale.

In an additional study, Rathee et al. [85] used multiview distance metric learning to learn a distance metric between the feature descriptors: Gabor features, Hog features, and LBP for detecting pain presence and pain intensity in 4 levels using Support Vector Machines. In [86], Gabor Filters were applied to the images and Principal Component Analysis (PCA) was implemented for feature compression. A linear SVM was used for the classification of four intensity levels. Zhao et al. [87] extracted facial landmark points, local binary pattern (LBP), and Gabor wavelet coefficients and trained an ordinal support vector regression (OSVR) for a 6-class classification task. An efficient optimization algorithm based on the Alternating Direction Method of Multipliers (ADMM) is proposed in order to solve the optimization problem that arises with parameter learning. Later, Shier and Yanushkevich [88] categorized pain into 3 classes using Gabor features as inputs to One-Against-One SVM.

Deep Learning Approaches

Deep learning methods, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have been widely proposed in the literature for facial pain estimation and have been shown to be effective in improving the accuracy and reliability of the estimation task.

Semwal et al [13] proposed a convolutional neural network architecture, which utilizes only three convolutional layers for three-class pain classification. Tavakolian et al. [89] employed a binary representation of facial expressions. A given face video is divided into non-overlapping segments of the same size. Then, they extracted features from each segment utilizing a Convolutional Neural Network (CNN). Low-level visual patterns and high-level structural information are combined using statistical methods to form the obtained features. The features are then converted into one binary code using a deep neural network. The classification of four pain levels was based on the Hamming distance. Furthemore, Ghazal Bargshady et al. [90] proposed a system consisting of the pre-trained Visual Geometric Group Face Convolutional Neural Network (VGGFace

CNN) and two different Recurrent Neural Networks (RNN) to classify pain into four categories.

In a similar fashion, Bargshady et al. [91] proposed utilizing the pre-trained VGGFace CNN to extract features. A hybrid deep model with two-stream convolutional neural networks associated with long-term and short-term memory (CNN-BiLSTM) was trained to predict four different levels of pain. Later, Tavakolian et al. [92] proposed a new similarity function for learning generalized representations with a Siamese network. To estimate pain, the learned representations are fed into a pre-trained CNN (VGG16, Inception-V1, ResNet-18, and ResNet-50).

Haarcascade Frontal Face Detector was used in [93] to detect faces from images, followed by image pre-processing steps. Images were fed into a CNN model based on a modified version of the VGG16 architecture to detect the pain's presence. In [94] popular Convolutional Neural Network architectures were compared, such as MobileNet, GoogleNet, ResNeXt-50, ResNet18, and DenseNet-161. These networks are utilized in two distinct modes: standalone mode and feature extractor mode. In standalone mode, the models are utilised to assess pain directly. In feature extractor mode, the middle layers' features are extracted and fed into SVR and RFR (Random Forest Regression). Huang et al. [95] presented a hybrid network (Hybnet) based on the extraction of multidimensional image attributes to evaluate pain. Spatiotemporal features are obtained using 3D CNN, spatial features are obtained using 2D CNN, and geometric information from facial landmarks is obtained using 1D CNN. The process of regression is accomplished using the fusion of the features.
Figure 3.1 displays the distinct groups of PSPI intensities observed in the literature.

Paper	[71]	[72]	[73]	[75]	[77]	[91]	[87]	[12]	[82]	[84]	[78]	[90]	[89]	[79]	[86]	Proposed Work
Classifier	SVM	SVM	SVM, DT, RF, 2NN	SVM	SVM	VGG16, Inception-V1, ResNet-18, ResNet-50	SVM	CNN	SVM	SVM	SVM	VGGFace-CNN,CNN-BiLSTM	VGGFace-CNN, RNNs	RVR	OSVR	SVM, DT, RF, ResNet18, ResNet50, CNN
Features	S-PTS, S-APP, C-APP	S-PTS, S-APs	PHOG, PLBP	LBP-TOP, LPQ-TOP, BSIF-TOP	P-HOG, HOG-TOP	Images	Gabor	Images	Spatiote-mporal Features	Gabor, Hog, LBP	CAPP, Log-Normal Filters	Images	Images	landmarks , DCT, LBP	LBP, Gabor	Features of 40 triangles, distances , Images
PSPI																
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																

Figure 3.1: Groups of PSPI intensities proposed in the literature

4 Methodology

4.1 Dataset

Participants

A total of 129 participants (63 men and 66 women) who self-reported experiencing shoulder pain issues participated in the initial study [49]. The sample was ethnically diverse, reflecting the Canadian metropolitan community from which it was recruited. Seventy percent of the participants had visited a doctor or physiotherapist for shoulder pain and had been given a diagnosis. Arthritis, bursitis, tendonitis, subluxation, rotator cuff injuries, impingement syndromes, bone spur, capsulitis, and dislocation were among the many diagnoses that were made. Sixty-five patients were admitted to using medications for their pain.

Apparatus and materials

The participants underwent testing in a lab room equipped with a bed for passive range-of-motion tests. For the purpose of capturing participant expressions, two Sony digital video cameras were used.

Procedure

Eight standardized range-of-motion tests involving the arm's abduction, flexion, internal rotation, and external rotation were performed on the participants. Abduction movements involved lifting the arm sideways and up in the frontal plane. The arm is raised up and forward in the sagittal plane during flexion. In internal rotation, the arm was abducted 90 degrees, twisted internally, and had its elbow bent 90 degrees. The only difference in the external rotation was that the arm was rotated externally. Abduction, flexion, internal and exterior rotations were assessed under both active and passive conditions. A physiotherapist carried out passive testing by moving the limb until the patient requested them to stop or the maximal range was reached. During passive tests, the patient was laying on the bed with a pillow supporting their head. Active tests, on the other hand, required the patient to actively participate by attempting to complete the full range of motion of the afflicted limb. The patient was standing during these tests.

To achieve a within-subject control, tests were performed on both the injured and unaffected limbs. Passive tests were taken from above at around a 70-degree angle to the participant's body plane, whereas active tests were captured from the front.

Only 200 sequences from 25 different subjects in the active condition of the UNBC-McMaster Pain Shoulder Archive are made available to the research community. To provide the highest possible accuracy, three distinct types of assessments were used to grade the sequences observed and the pain experienced by the patients. Participants filled out the three scales after every test :

1. **Affective-motivational (AFF) scale and Sensory scale (SEN) :** Participants provided verbal pain ratings for each test using a card with verbal pain descriptions. Each card had two descriptive scales derived from Heff et al. [50]. The first group of words represented the sensory dimension (intensity) of pain. The second component was comprised of terms that reflected the affective-motivational dimension (unpleasantness). Each scale contained 15 items, labeled from "A" to "O." The affective-motivational (AFF) scale began at "bearable" and ended at "excruciating.", while the sensory scale (SEN) ranged from "extremely weak" to "extremely intense."

2. **Visual Analog Scale (VAS) :** Participants rated each test using a set of 10 cm line Visual Analog Scale (VAS) with two endpoints representing 0 ('no pain') and 10 ('pain as bad as it could possibly be').
3. **Observer Pain Intensity (OPI) :** An independent observer trained to identify pain expressions assigned an Observer Pain Intensity (OPI) rating to each sequence on a 6-point scale from 0 (no pain) to 5 (severe pain).

A total of 48,398 FACS-coded and AAM - tracked frames were extracted from these 200 sequences. All of the frames in this dataset were coded using the PSPI. For the purpose of this study, only PSPI scores are utilized.

4.2 Multiclass Imbalanced Classification Problem Analysis

Every frame in the database was coded according to its PSPI score on a scale from 0 to 15. The database contains 200 sequences from 25 different subjects, for a total of 48,398 image frames. According to the Table 4.1, the number of images with no pain PSPI score labels is higher than the number of images with other labels, and there are only 322 images in this database with PSPI labels higher than 6. As a result, given the unique characteristics of the database, it is probable that any model could be biased toward the prediction of no pain at the risk of missing pain frames.
In our research, we categorized the pain intensities into 3 classes:

No-pain: PSPI values equal to 0

Weak pain: PSPI values equal to 1 and 2

Strong pain: PSPI values greater than or equal to 3

In machine learning, multiclass classification refers to the process of classifying instances or samples into one of three or more classes based on the data (binary classification refers to the process of classifying instances into one of two classes). Multiclass classification assumes that each sample is assigned with only one label. From Figure 4.1 it is observed that the dataset is not homogeneous and classes with and without pain, are not equally represented in the dataset. In particular, 40029 frames (82.71%) belong to the no–pain class, 5260 frames (10.87%) to the weak pain class, and 3109 frames (6.42%) to the strong pain class. Figure 4.2 shows the number of frames captured from each patient.

PSPI Score	Frequency
0	40029
1-2	5260
3-4	2214
5-6	512
7-8	132
9-10	99
11-12	124
13-14	23
15-16	5

Table 4.1: The inventory on the UNBC-McMaster Shoulder Pain Archive according to the Prkachin-Solomon Pain Intensity (PSPI) pain metric, where the frequency of each pain intensity is given [14].

Applications in the real world are experimenting with skewed data. Problems having a skewed distribution of class cardinalities, in which one or more classes (minority classes) are significantly underrepresented in comparison to the others (majority classes), are characterized as imbalanced. Depending on the arrangement, a multi-class imbalanced dataset may have one minority class and numerous majority classes, or one majority class and several minority classes. Specific to their rarity, minority classes often play a crucial role in a given task and deserve to be properly recognized.

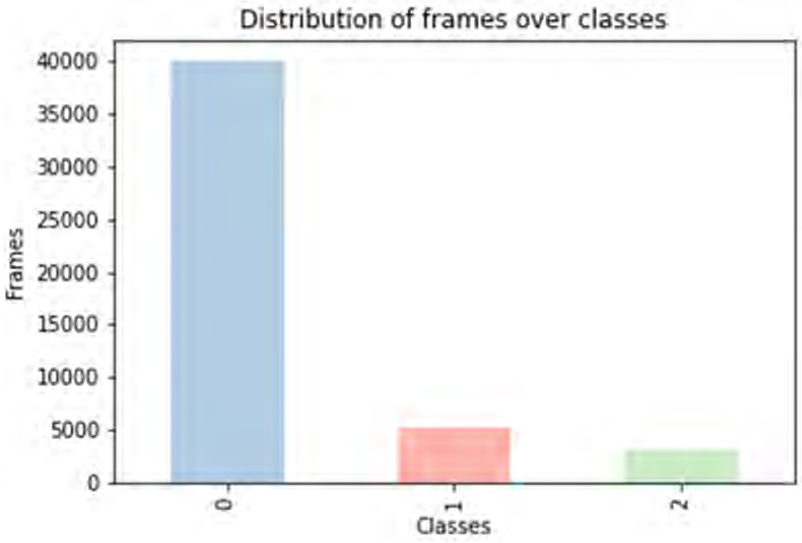


Figure 4.1: Distribution of frames over classes

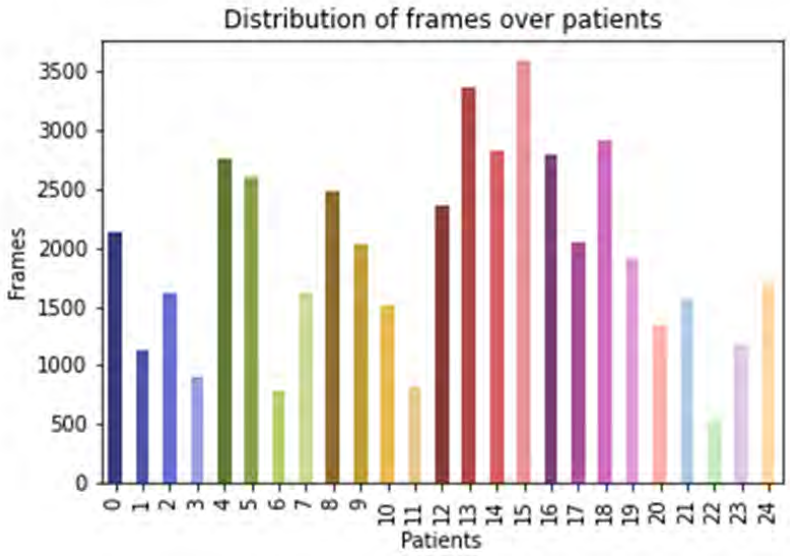


Figure 4.2: Distribution of frames over classes

4.3 Features

4.3.1 Active Appearance Models

In the UNBC-McMaster Shoulder Pain Dataset, all frames have their own AAM features which comprise 66 facial landmarks. In the field of computer vision, Active Appearance Models are an extensively employed technique for landmarks localization. The Active Appearance Model (AAM) was first introduced by Taylor and Cootes in 2001 [96]. The basic objective is to model the shape and appearance of individuals or objects. This can be accomplished by aligning a source image containing the object of interest with a predefined linear shape model that exhibits linear appearance variation.

The shape s of an AAM is defined by the vertex locations of a 2D triangulated mesh [14], [97]. An example of a mesh can be seen in Figure 4.3. Based on the coordinates of the n vertices, an AAM's shape can be mathematically described as:

$$s = [x_1, y_1, x_2, y_2, ..., x_n, y_n] \tag{4.1}$$

Due to the fact that AAMs allow variations in linear form, the shape s can be expressed as a base shape s_0 plus a linear combination of m shape vectors s_i [97]:

$$s = s_0 + \sum_{i=1}^{m} p_i s_i \tag{4.2}$$

, where the coefficients p_i represent the shape parameters.

The shape parameters can be split into rigid similarity parameters $\mathrm{p_s}$ and non-rigid object deformation parameters $\mathrm{p_o}$, such that $p^T = [p_s^T, p_o^T]$ [14]. Similarity parameters are related to geometric transformations such as translation, rotation, and scale while the residual parameters indicating non-rigid geometric transformations are linked to the determining object shape like, mouth opening, eyes shutting and so forth. AAMs are typically calculated from training data by performing Principal Component Analysis (PCA) on the meshes [97]. The base shape s_0 is considered the mean shape. The most popular technique for obtaining facial landmarks is gradient descent [98]. Each video sequence's keyframes on the McMaster UNBC Pain Dataset were manually identified, and the remaining frames were automatically aligned utilizing the gradient descent AAM fitting algorithm proposed in [97].

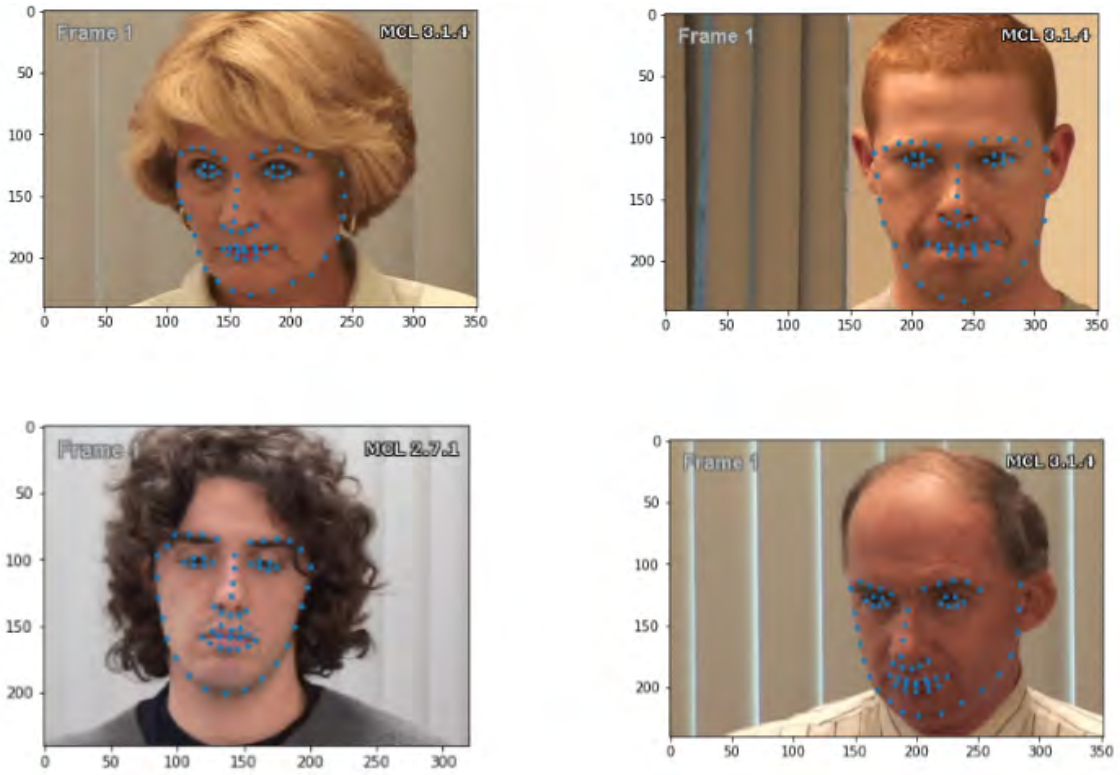


Figure 4.3: The 66 AAM tracked landmarks on randomly selected frames

4.3.2 Delaunay Triangulation

The Delaunay Triangulation (DT) of a point set V, proposed by Boris Nikolaevich Delone [99] in 1934, is a triangulation of V with well-shaped triangles [100]. It is distinguished by the circumcircle property, which states that no triangle has a circumscribing circle that encloses any point in V [10].

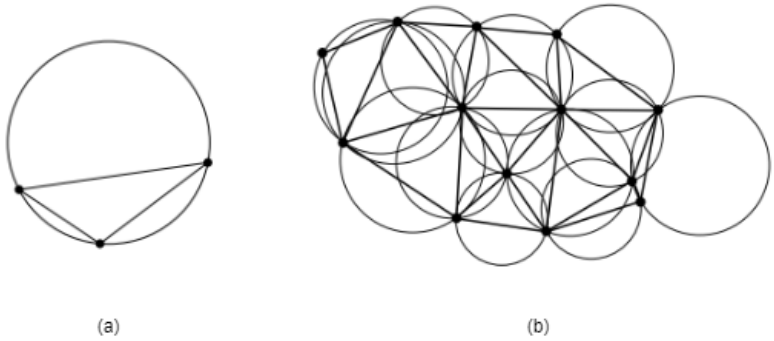


Figure 4.4: (a) Circumcircle of a triangle, [9]
(b) Every triangle in a Delaunay triangulation has an empty circumcircle, [10]

Definition 1: A triangle's circumcircle, also known as circumscribing circle, is the only circle that crosses all three of its vertices (Figure 4.4(a)). An edge's circumcircle is any circle that crosses through both of its vertices. An edge in the plane has an endless number of circumcircles, whereas a triangle only has one [10].

Definition 2: In the setting of a finite point set V, a triangle is Delaunay if its vertices are in V and its circumcircle is empty: there is no other point from V inside. Note that any number of points may lie on the circumcircle of a Delaunay triangle. If an edge contains at least one empty circumcircle and has vertices in the V, it is considered to be Delaunay [10]. As shown in figure 4.4(b), a DT of V is a triangulation of V in which each triangle is a Delaunay.

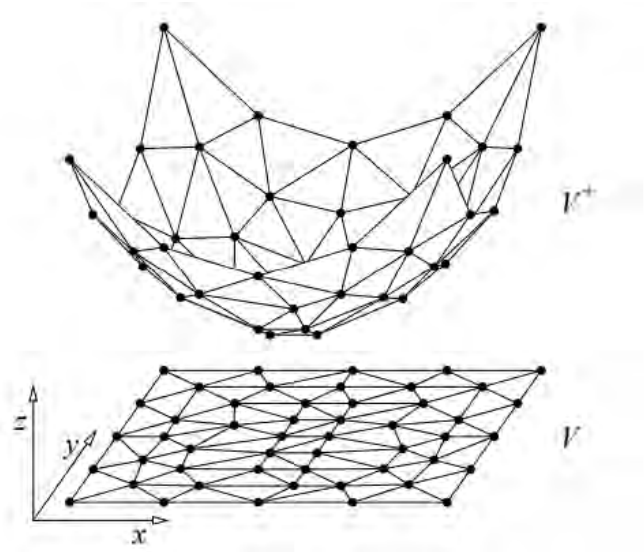


Figure 4.5: The parabolic lifting map, [10]

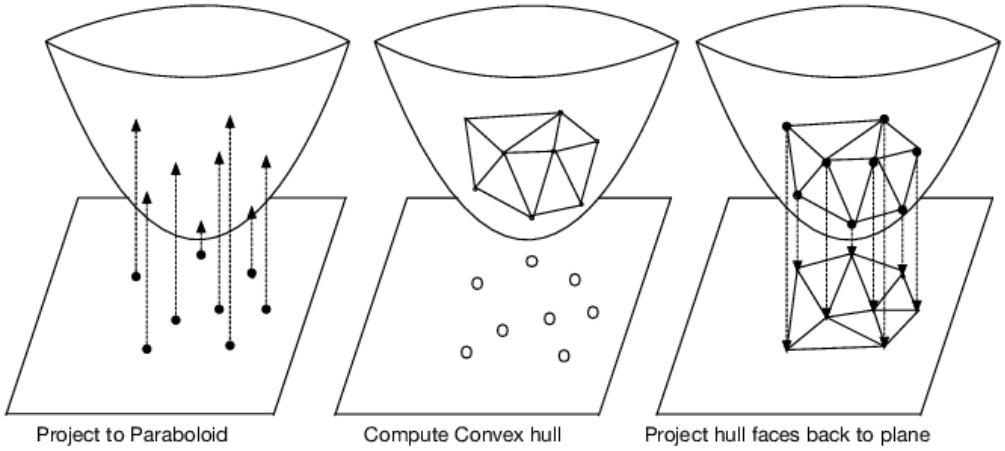


Figure 4.6: Computation of Delaunay triangulation through convex hull, [11]

The construction of the 2-dimensional DT is equivalent to the construction of the convex hull of the lifted set of points in 3 dimensions due to the fact that convex hulls are simpler to create [101]. The basis of the connection is the paraboloid $z = x^2 + y^2$.

Each point $p = (x, y)\ \epsilon\ E^2$ maps to a point $p^+ = (x, y, x^2 + y^2)\ \epsilon\ E^3$ through the parabolic lifting map. p^+ is named as the lifted companion of p [10]. Be aware that the paraboloid in three dimensions defines a surface whose vertical cross sections are parabolas and whose horizontal cross sections are circles [101].

Consider the convex hull $conv(V^+)$ of the lifted points $V^+ = v^+ : v\ \epsilon\ V$. The connection is as follows: every facet of the lower envelope of $conv(V^+)$ projects to a simplex of the Delaunay triangulation of V [101]. The lower convex hull of $conv(V^+)$ is the portion of the convex hull that is visible to a viewer standing at $z = -1$. A face f of $conv(V^+)$ is considered downward-facing if no point in $conv(V^+)$ is directly below any point in f with respect to the z-axis. The collection of downward-facing faces is referred to as the underside of $conv(V^+)$. The Delaunay subdivision of V is obtained by projecting the underside of $conv(V^+)$ to the x-y plane without using the z-coordinates of any of the points [10]. The downward-facing faces are shown in figure 4.5.

To summarize,, consider (p, q, r) to be elements of V, and p_0, q_0, r_0 to be the projections of these points onto the paraboloid. Then $p_0q_0r_0$ define a face of the lower convex hull of $conv(V^+)$ if and only if (pqr) is a triangle of the DT of V. The process is shown in Figure 4.6. The objective of triangulation is to generate a mesh. A DT takes a set of points as input and produces a set of triangles as output [102].

4.3.3 Triangle Analysis

A set of three distinct points $A(x_1,y_1)$, $B(x_2,y_2)$ and $C(x_3,y_3)$ defines a triangle, such as in Figure 4.7(a). Based on the points' coordinates, the area of the triangle is given by:

$$Area = \frac{1}{2}(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)) \tag{4.3}$$

The three points can also be represented as vectors as illustrated in Figure 4.7(b) where,

- $\overrightarrow{BA} = B - A = (x_2 - x_1, y_2 - y_1)$
- $\overrightarrow{BC} = C - B = (x_3 - x_2, y_3 - y_2)$

The scalar product of the two vectors is defined as:

$$\overrightarrow{BA} \cdot \overrightarrow{BC} = \left\|\overrightarrow{BC}\right\| \left\|\overrightarrow{BA}\right\| \cos(\theta) \tag{4.4}$$

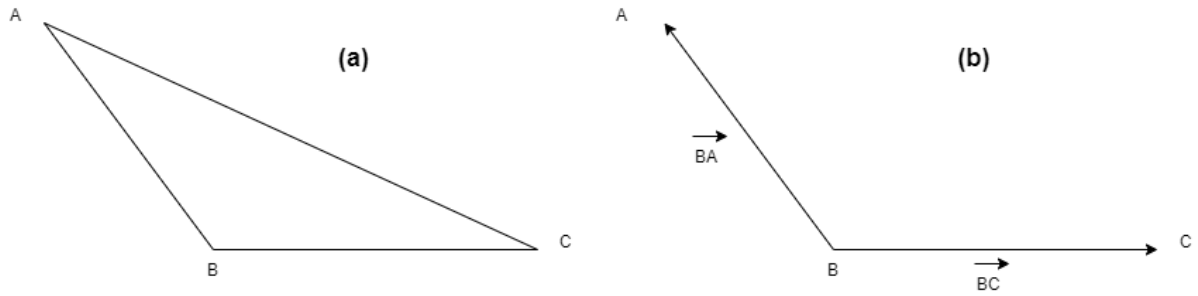


Figure 4.7: (a) Triangle ABC (b) Angle between vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$

,where $\| \|$ is the magnitude of the vector and θ is the angle between the two vectors. For example, the magnitude of the vector $\overrightarrow{BA}$ is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
The angle θ can be calculated as:

$$\theta = \cos^{-1}\left(\frac{\overrightarrow{BA} \cdot \overrightarrow{BC}}{\left\|\overrightarrow{BC}\right\| \left\|\overrightarrow{BA}\right\|}\right) \tag{4.5}$$

Angle θ corresponds to the angle $A\widehat{B}C$ of the triangle ABC. In the same approach, we could define the angles $B\widehat{A}C$ and $A\widehat{C}B$.

4.3.4 Geometric Features

We trained Machine Learning algorithms with spatial features extracted from the frames. The rationale behind the use of geometric features is that they are based on the facial features' shapes, which reveal information about the expression. The majority of facial muscle activity results in the displacement of geometric facial features derived from the location of facial landmarks. Facial movements, for example, can raise or lower the outer corner of the brows or may extend or shorten the mouth.

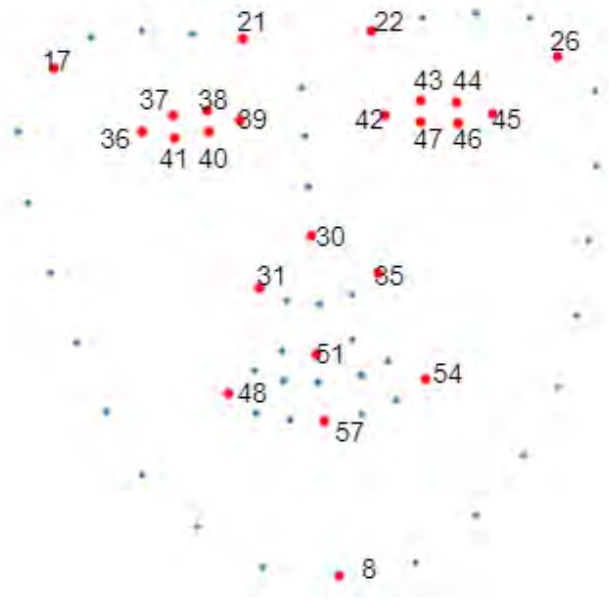


Figure 4.8: The 24 selected landmarks

As a reference image, we selected the image in Figure 4.9(a). To keep the dimensionality of the feature vectors lower, we used a smaller set of facial landmarks. More specifically, we selected 24 out of the 66 facial landmarks provided by the McMaster UNBC Pain Dataset for further analysis, as demonstrated in Figures 4.8, 4.9(b).

Following that, we implemented the Delaunay Triangulation giving as input the 24 FCPs, and a group of 40 triangles was created. The Delaunay triangulation is calculated using the spatial.Delaunay function in the scipy Python library. The Delanaunay triangles are drawn in Figures 4.9(c), (d). Having information on the vertices that belong to each of the 40 simplices, we found the triangles on each image. For example, in the Figure 4.10 we can notice the triangle produced by the FCPS 39, 21 and 22 in the reference image and in another patient's image. The coordinates of the three points differ in the two images.

The feature extraction procedure for each triangle follows the algorithm described in Algorithm 1. Due to the huge variation between the images, some of them do not have all the triangles. We incorporated these exceptions in our implementation by performing a collinearity check. Specifically, three points are referred to as collinear when they lie on the same straight line. Therefore, if we try to compute the area based on the coordinates of collinear points, the result will be zero. In this scenario, we set one of the three angles to 180° and the other ones as 0.

Figure 4.9: (a) Reference Image (b) Locations of the 24 landmark points (c)Delaunay Mesh on the reference Image (d)Delaunay Mesh on another patient

Figure 4.10: (a) Triangle produced by FCPs 11, 2 and 3 in Reference Image (b) Triangle produced by FCPs 11, 2 and 3 in another patient's image

Let A(x_1,y_1), B(x_2,y_2) and C(x_3,y_3) be the three vertices that are entered into the Algorithm 1. For each image in the dataset, the following geometric features are obtained:

- Feature vector of 40 Maximum angles (one maximum angle for each triangle),
- Feature vector of 40 Minimun angles (one minimum angle for each triangle),
- Feature vector of 40 Areas(one area for each triangle),
- Feature vector of the Euclidian Distances between all pairs of the 24 facial characteristic points, excluding self-to-self pairs. The size of the feature vector is 276. Distances are calculated using the cdist function in the scipy.spatial.distance module.

Algorithm 1 Feature extraction procedure for triangle

Input: A, B, C
Output: $Maximum\,angle$, $Minimum\,angle$, $Area$
Compute the $Area$ of the triangle as in 4.3.3
if $Area = 0$ **then**
  $A\widehat{B}C = 180°$
  $B\widehat{A}C = 0$
  $A\widehat{C}B = 0$
else
  Compute $A\widehat{B}C$ and $B\widehat{A}C$ as in 4.3.3
  $A\widehat{C}B = 180°$ - $A\widehat{B}C$ - $B\widehat{A}C$
end if
$MaximumValue = \max(A\widehat{B}C, B\widehat{A}C, A\widehat{C}B)$
$MinimumValue = \min(A\widehat{B}C, B\widehat{A}C, A\widehat{C}B)$

4.3.5 Image-Based Features

In order to transform the images into a more suitable format for classification using deep learning models, the following preprocessing methods have been utilised in this project:

1. **Image Alignment**

   In mathematic perspective, slope represents the steepness of a straight line from the horizontal plane and is commonly denoted by m. It is also often referred to as the gradient of a line or the angular coefficient of a line as it represents the angular incline from the horizontal plane. A steeper line results from a larger value of the slope. The slope is essentially the change in height over the change in horizontal distance. Linear functions are often represented algebraically by : $y = mx + b$ or $ax + by = c$ [103]. In the general case, if we take two points $A(x_1, y_1)$ and $B(x_2, y_2)$, the slope is calculated as:

   $$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.6}$$

   In the equation above, $y_2 - y_1 = \Delta_y$, or vertical change and $x_2 - x_1 = \Delta x$, or horizontal change, are visualised in the Figure 4.11. In trigonometry, the tangent function $m = \tan\theta$ relates m to its angle of incline θ [103].

   Consequently, the angle θ made by the line produced by the points A and B with the horizontal x-axis is defined as:

   $$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{4.7}$$

   In our approach, for each image, we calculated the angle θ between the line connecting the right and the left corner of the eyes and the horizontal axis. The alignment procedure is based on the rotation of the image around its center by the angle θ. The Rotation matrix applied to each image is described as:

   $$M = \begin{bmatrix} \alpha & \beta & (1-\alpha)c_x - \beta c_y) \\ -\beta & \alpha & \beta c_x + (1-\alpha)c_y \end{bmatrix} \tag{4.8}$$

   where,

   - $\alpha = scale\, cos(\theta)$
   - $\beta = scale\, sin(\theta)$
   - c_x and c_y are the coordinates of the center of the image

   An example of an Aligned image is shown in Figure 4.13(b).

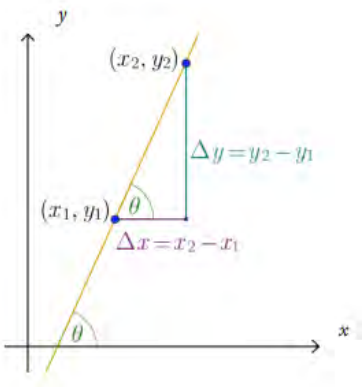


Figure 4.11: Slope

Figure 4.12: Boundary Points

2. **Face Detection and Image Cropping**

   Face Detection was carried out using the coordinates of the boundary points of the face. An example of the boundary points can be seen in Figure 4.12. Specifically, the 4 points used are:

   - Point 0 : It has the minimum x value on the horizontal axis
   - Point 16: It has the maximum x value on the horizontal axis
   - Point 19: It has the minimum y value on the vertical axis
   - Point 8 : It has the maximum y value on the vertical axis

   The steps of face detection and image cropping are demonstrated in Algorithm 2. Briefly, we compute the *height* and the *width* of the face. The *height* describes the vertical distance between the eyebrows and the chin. During pain exposure, formations are also observed on the forehead. We notice that the distance between the eyebrows and the top of the forehead is one-quarter of the distance between the eyebrows and the chin. In order to keep the forehead in the cropped image, we compute a new height that equals to $height + \frac{1}{4}height$. Finally, we crop the image in a rectangular size. The output of this procedure is a cropped image that includes only the face of the patient as illustrated in Figure 4.13(c).

3. **Resizing**
   The NxN cropped images have different sizes. Therefore, the images are resized to the same dimensions 224×224 to adapt as inputs to the CNN models.

4. **Normalization**
   The range of pixel intensity values is changed to [0, 255].

Algorithm 2 Face Detection and Image Cropping

Input: Point 0, Point 8, Point 16, Point 19
Output: Cropped Image

min_x = X value of Point 0
max_x = X value of Point 16
min_y = Y value of Point 19
max_y = Y value of Point 8
Compute the Height : $Height = max_y - min_y$ {Distance between eyebrows and chin.}
Compute the distance between eyebrows and the top of the forehead: $Quarter = \frac{1}{4} Height$
Compute the Final Height: $Final\,Height = Height + Quarter$
Compute the Width: $Width = max_x - min_x$
Compute the difference between Height and Width: $Difference = Final\,Height - Width$
if $Difference > 0$ **then**
    $Top = min_y - Quarter$
    $Bottom = max_y$
    $Left = min_x - Difference/2$
    $Right = max_x - Difference/2$
else
    $Top = min_y - Quarter + Difference/2$
    $Bottom = max_y - Difference/2$
    $Left = min_x$
    $Right = max_x$
end if
Crop the image in the boundaries: Top, $Bottom$, $Left$ and $Right$

Figure 4.13: (a) Initial Image (b) Aligned Image (c) Cropped Image containing only the face (d) Resized image to 224 × 224 (e) Normalized Image

4.3.6 Histogram of Oriented Gradients

Besides the geometric features that utilize facial fiducial points to describe the shape of the face, extraction of appearance features is also popular in facial expression analysis. Appearance features are primarily used to describe the texture of the face. In our research, Histogram of Oriented Gradients (HOG) is employed.

The HOG detector was introduced by Dalal and Triggs [104] in 2005. It is a feature descriptor that examines how gradients are distributed throughout an image. The histogram is determined by the gradient direction and magnitude. The magnitude specifies which bin to fill, and the direction specifies how much to fill it. Detecting the gradient is useful in identifying and localizing objects by calculating features such as texture, edges, and corners, which differentiate objects from their surroundings and reveal their boundaries. [105]. For example, if we are interested in facial expressions, HOG can detect the location and angle of the mouth.

The aligned and cropped images produced in the section 4.3.5, were resized from dimensions 224 × 224 to 128 × 128 and converted to grayscale. HOG descriptors can be generated by dividing the image into small connected regions known as cells. Then, for each cell a histogram of oriented gradients is computed for all pixels of the cell. The HOG algorithm consists of four steps, namely: gradient computation, orientation binning, block normalisation and object detection.

Gradient Computation
Horizontal (G_x) and vertical (G_y) image gradients are calculated by filtering the image through Sobel kernels. To obtain the gradient images, the convolution operation is applied:

$$G_x = I \star D_x \tag{4.9}$$

$$G_y = I \star D_y \tag{4.10}$$

,where I is the input image, D_x and D_y are the filtering masks $[+1, 0, -1]$ in the x-direction and in the y-direction respectively.
Then, the magnitude and the angle of the gradients can simply be computed by using the following formulas:

$$Magnitude(m) = \sqrt{G_x^2 + G_y^2} \tag{4.11}$$

$$Angle(\theta) = \left| \tan^{-1}(\frac{G_y}{G_x}) \right| \tag{4.12}$$

Weighted votes in each cell
The size of the cell is set to 8×8 pixels, leading to (128/8 = 16) 16x16 = 256 cells. Each pixel of a cell gives a weighted vote to the histogram. The weight of the vote is simply the gradient magnitude at the given pixel.

Orientation Binning
Generally, the number of orientations controls the number of bins in the histogram. In our implementation, histograms of edge gradients with 9 orientations are calculated from each of the local cells. A histogram is compiled in the range of 0-180 degrees with the bin width equal to 20.

Block Normalisation
The gradient values must be normalized locally to account for changes in illumination and contrast. This involves combining the cells into blocks. Because these blocks commonly overlap, each cell contributes to the final feature vector multiple times. The gradient histograms of each cell in the current block are concatenated before normalizing the feature vector with L1 or L2. After normalizing all blocks, histograms are concatenated to generate a feature vector. The block's size is set to 16×16 pixels, generating a HOG feature vector that is comprised of 9x16x16 = 2.304 features.

4.4 Dataset Splitting

One of the most important parts of Machine Learning is building models that can predict and generalize well. At the end of training, the model should not only be able to accurately predict results for training input samples, but also to perform well on data that it has never seen before. The ability to generalize depends on how complicated the model is. The complexity of a machine learning model is determined by its hyperparameters [106]. Frequently, a trained model achieves highly performance on the data used for training but poorly performance on patterns that were not included in the training data. This phenomenon is called over-fitting.

It is essential to divide the data into a training set and a validation set in order to find the optimal set of model parameters that reaches a satisfying balance between generalization and complexity [107], [106]. In addition, when executing a learning algorithm over a training set, changing the hyperparameter settings can result in different models. Model selection is the process of selecting the top-performing model from a set of models that were created using different hyperparameter settings based on model's performance to the validation set.

Recent studies have demonstrated that the validation set alone may not always be adequate to determine how well a model is performing [106]. Additionally, a study by Harrington et al. [108] showed that having only training and validation sets can lead to inaccurate estimations of predictive performance. These findings highlight the need for another independent test set. By using this test data after training, the model's ability to generalize on unseen data can be measured [106]. Furthemore, according to [109], cross-validation can also produce optimistic results.
In our study, we implemented different dataset-spitting techniques:

1. PATIENT – WISE SPLITTING

   In this approach, we considered the need to keep all data of each patient into the same subset and the labels' distribution in the subsets as close as possible to the distribution of the labels in the initial dataset, in order to split the dataset.

   General random data splitting does not guarantee that all the minority classes will be represented in the test set. As a result, stratifying the dataset is a plausible approach for our problem. To maintain the original class proportion in the subsets, stratification divides a dataset so that each class is accurately represented in the subsequent subsets.

   Patient-wise splitting is beneficial in medical research and other applications that collect data from individual patients. By guaranteeing that there are no overlapping patients between the training and test sets, the performance of the classifier is evaluated on patients whose data has not been utilized for training. Therefore, the model is less likely to memorize the training data and perform poorly on unseen data.

   Two different ways of patient-wise splitting are examined:

   (a) **Patient-Wise Splitting 1** : Stratified Splitting of the dataset into 60% training set, 20% testing set, and 20% validation set.
   When dividing data into three sets, it's essential to ensure that the training, validation, and test sets are all representative of the whole data set. Therefore, stratifying target labels increase stability by improving the representation of minority classes in each split [110]. Having a training-validation pair for hyperparameter tuning and model selection enables us to maintain the independence of the test set for model evaluation [111].

Table 4.2: Patient-Wise Splitting 1

	Total frames	No-Pain frames	Weak-Pain frames	Strong-Pain frames
Whole Dataset	**48,398**	**40,029** (/48,398 = 82.71%)	**3,109** (/48,398 = 10.87%)	**5,260** (/48,398 = 6.42%)
Training Set	**29,917** (/48,388 = 60%)	**24,714** (/29,917 = 82.61%)	**3,306** (/29,917 = 11.05%)	**1,897** (/29,917 = 6.34%)
Validation Set	**9,628** (/48,388 = 20%)	**7,368** (/9,628 = 83,23%)	**889** (/9,628 = 11.04%)	**596** (/9,628 = 6.73%)
Test Set	**8,853** (/48,388 = 20%)	**7,947** (/8,853 = 82.54%)	**1,065** (/8,853 = 11.06%)	**616** (/8,853 = 6.40%)

Table 4.3: Patient-Wise Splitting 2

	Total frames	No-Pain frames	Weak-Pain frames	Strong-Pain frames
Whole Dataset	**48,388**	**40,029** (/48,398 = 82.71%)	**3,452** (/48,398 = 10.87%)	**5,260** (/48,398 = 6.42%)
Training Set	**32,490** (/48,388 = 70%)	**26,764** (/32,490 = 82.38%)	**3,452** (/32,490 = 10.62%)	**2,274** (/32,490 = 7.00%)
Test Set	**15,908** (/48,388 = 30%)	**13.256** (/15,908 = 82.54%)	**1,808** (/15,908 = 11.37%)	**835** (/15,908 = 5.25%)

The general steps of this method are:
Step 1: Splitting the dataset into three parts: Training set for model fitting, validation set for model selection, and test set for the final evaluation of the selected model.
Step 2: Hyperparameter tuning phase: Training lots of models with different hyperparameter settings on the training set, and evaluating the performance of each model using the validation set. Finally, a metric is used to compare the models and the optimal model with better performance on thε validation set is selected.
Step 3: The selected model is evaluated on the test set, in order to achieve an unbiased estimation of the generalization performance of the model.

In our case, the dataset consisting of 25 patients was split into training (13 patients), validation (6 patients), and testing (6 patients) subsets. A similar distribution of classes as in the original dataset was maintained in each subset, as presented in Table 4.2. In addition, the dataset of 48,398 images was divided with a ratio of 60:20:20 for training, testing and validation respectively.

(b) **Patient-Wise Splitting 2** : Stratified Splitting of the dataset into 70% training set, and 30% testing set. Validation is performed through cross-validation on the training set.

The initial dataset is split into two parts: (a) the training set and (b) the test set. The basic idea is to maintain an independent test subset that is not used during training and model selection.

The cross-validation method we implemented is the Stratified Group k-fold. It is a variation of k-fold cross-validation with the goal of ensuring that each fold contains roughly the same proportions of different target classes as the original dataset. For example, if the original dataset has a distribution of 60% class A and 40% class B, then each fold should also have a similar distribution of 60% class A and 40% class B. The data is first divided into k-folds, with each fold containing nearly the same proportion of samples from each class. If a patient is in fold k, then those patient's samples will not appear in fold k-1. With k equal to 3, three iterations are performed. The final results are obtained by taking the average performance of the model across all k iterations.

The general steps of this method are:
Step 1: Splitting the dataset into into two parts: Training set for model fitting and test set for the final evaluation of the selected model.
Step 2: Hyperparameter tuning phase: The Stratified-group k-fold cross-validation is applied to the training set for each hyperparameter configuration, resulting in many models and performance estimates. The hyperparameter setting that produced the best results is selected.
Step 3: Training the best model with the whole training subset and obtaining the final model.
Step 4: Evaluation of the final model with the test set.

In this approach, the dataset consisting of 25 patients was initially split into training (17 patients), and testing (8 patients) subsets. A similar distribution of classes as in the original dataset was maintained in each subset, as presented in Table 4.3. In addition, the dataset of 48,398 images was divided with a ratio of 70:30 for training and testing respectively.

Figures 4.14, 4.15 visualize the separation of patients and their frames into subsets. Specifically, in the "Patients" line, the 25 patients are shown arranged next to each other, and in the "Frames" line, their frames are shown arranged in a row, colored according to the class they belong to. The "Splitting" line describes which patients along with all of their frames belong to each set. In Figure 4.16, the cross-validation performed on the training set of the Patient-Wise Splitting 2 is visualized. Specifically, in each iteration (0, 1 and 2), it is displayed which patients belong to the training set and which to the validation set.

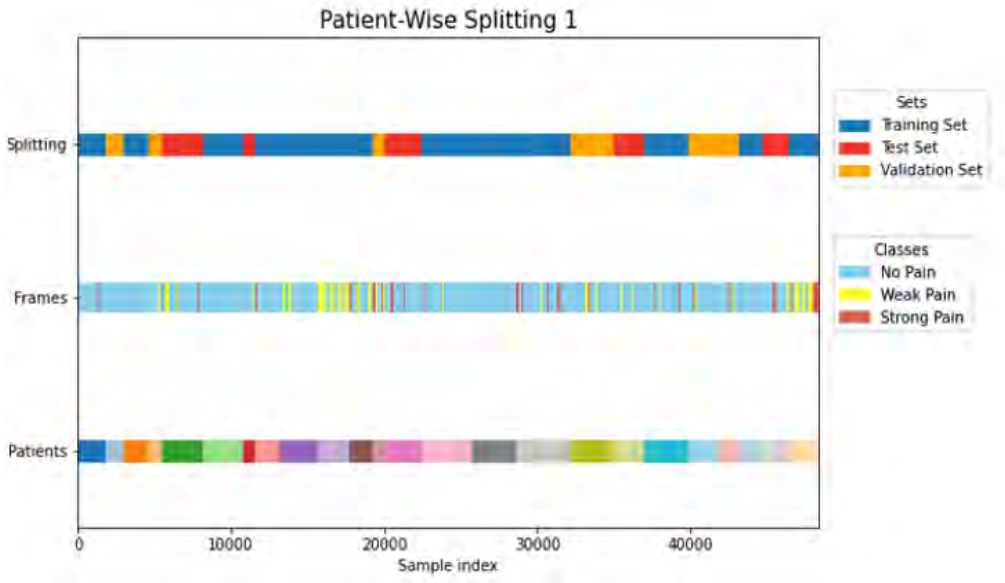


Figure 4.14: Patient – Wise Splitting 1

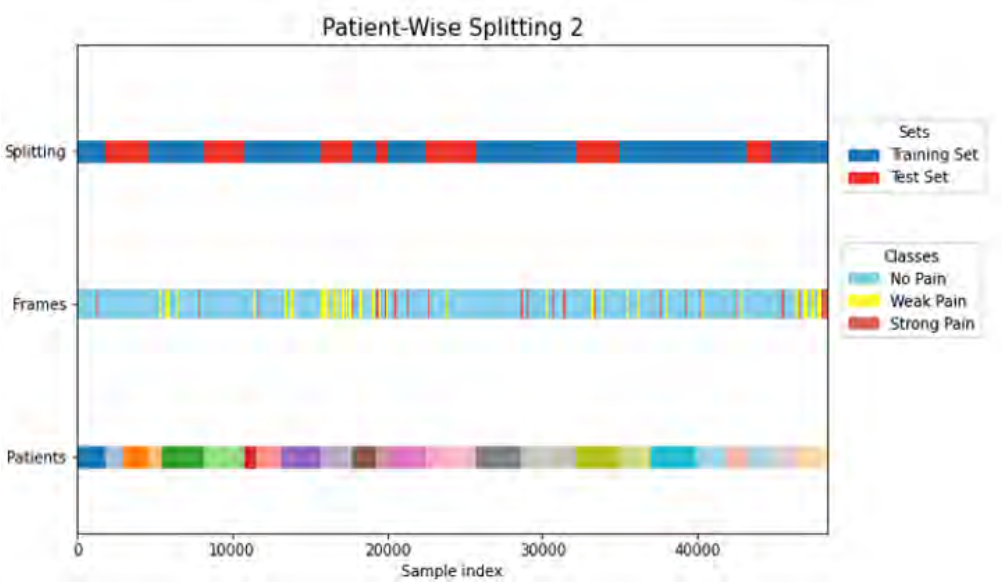


Figure 4.15: Patient – Wise Splitting 2

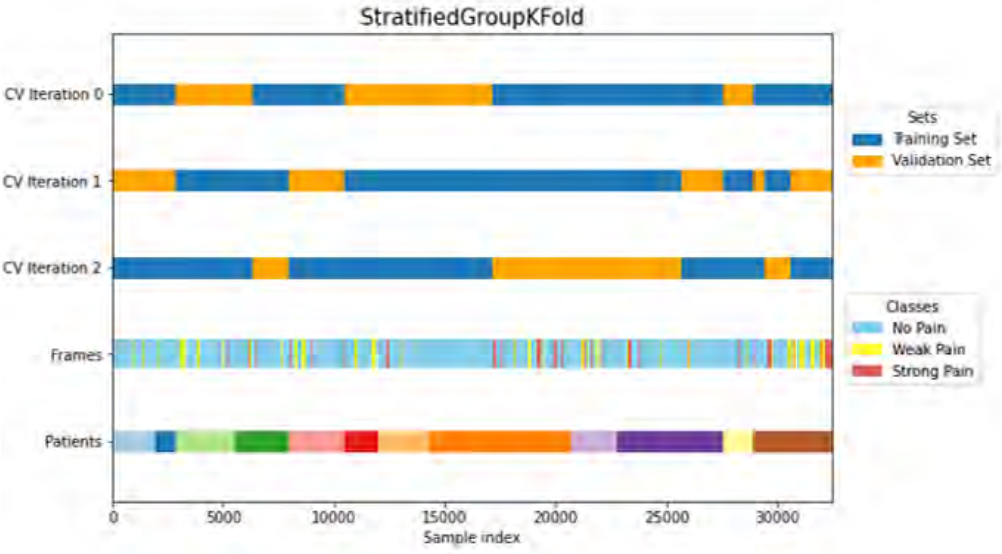


Figure 4.16: Stratified-Group k-fold Cross Validation

Table 4.4: Sequence-Wise Splitting

	Total frames	No-Pain frames	Weak-Pain frames	Strong-Pain frames
Whole Dataset	**48,398**	**40,029** (/48,398 = 82.71%)	**3,109** (/48,398 = 11.03%)	**5,260** (/48,398 = 6.92%)
Training Set	**26,730** (/48,388 = 60%)	**21,932** (/26,730 = 80.38%)	**2,948** (/26,730 = 11.03%)	**1,850** (/26,730 = 6.92%)
Validation Set	**8.902** (/48,388 = 20%)	**7,155** (/8.902 = 80.3%)	**1,131** (/8.902 = 12.71%)	**616** (/8.902 = 6.92%)
Test Set	**9,947** (/48,388 = 20%)	**8,123** (/9,947 = 81.66%)	**1,181** (/9,947 = 11.87%)	**643** (/9,947 = 6.46%)

2. SEQUENCE – WISE SPLITTING

   In this splitting, we considered the need to allocate all data of each sequence into the same subset and the labels' distribution in the subsets as closely as possible to the initial dataset in order to split the dataset. However, sequences from the same subject can be present in all the subsets. The performance of the classifier is evaluated on sequences whose data has not been utilized for training.

   The dataset containing 200 sequences was split into training (60%, 114 sequences), validation(20%, 39 sequences), and testing (20%, 38 sequences) subsets. A similar distribution of classes as in the original dataset was maintained in each subset, as presented in Table 4.4. In addition, the dataset of 48,398 images was divided with a ratio of 60:20:20 for training, testing, and validation, respectively.

3. LEAVE-ONE-SUBJECT-OUT CROSS VALIDATION -LOSO

   According to Leave-One-Subject-Out (LOSO), [110], the samples of one subject are omitted from the modelling process in order to generate a model from the data of the other subjects. The model is then tested on the samples of the subject who was previously omitted [112].

   Based on the study by Hawkins [113] LOSO is the most reliable method for testing a model with participant-level data. A LOSO technique is considered an optimal method of model evaluation when observing significant differences between participants. This is frequently the case with medical datasets, which are often characterized by limited individuals, and recurrent measurements, where each patient provides many samples [114].

   As illustrated in Figure 4.17, 25 iterations, are executed. In every fold, the model is trained on 24 patients and evaluated on the "left out" patient. As a subject may not contain painful frames, the test data occasionally featured with samples only from one class. However, since 24 of the 25 participants are included in the training set, the training data always includes examples from all the 3 classes.

   The importance of utilizing LOSO as the evaluation methodology is something we want to emphasize. According to [112], k-fold cross-validation findings indicate that the model tends to perform better when data from all subjects are used for training. In this instance, the training and test sets are not divided based on subjects, meaning that both the training and testing sets may contain data from each subject. This approach learns subject-specific features rather than pain-related features, and because in a more realistic case the objective is to categorize a new, unseen subject, we provide subject-independent findings using LOSO.

Figure 4.17: Leave-One-Subject-Out Cross Validation

4.5 Solutions to Imbalance Problem

The default behavior of an Machine Learning model when there is a class imbalance is to over-classify the dominant class. Although it may result in high accuracy, the model is biased, with a larger likelihood of misclassification for the minority class. Several approaches have been presented over the years to deal with the class imbalance problem, which can be generally divided into two main categories: **data level** and **algorithm level** methods.

Data-level approaches use resampling techniques to balance the data distribution on the training set. Approaches at the algorithm level modify the machine learning algorithm to accommodate imbalanced data. The adjustment is achieved using cost-sensitive methods that take into account different misclassification costs for each class or minimize the cost error rather than maximize the accuracy rate [115].

Oversampling techniques duplicate current samples or add more samples to the minority classes [116]. Therefore, they increase the time needed to complete the training due to the larger training data set [115]. On the other hand, Undersampling approaches remove samples from majority classes. As a result, the training procedure is simple to implement and addresses the run time and storage issues [117] The sampling procedures for multi-class datasets should be modified to account for the individual attributes of classes and their interrelationships [115]. Class decomposition schemes are another widely used approach to handle multi-class data [118]. The benefit of such approaches is the simplification of the problems into multiple two-class problems. One-versus-all (OVA) and one-versus-one (OVO) schemes are two of the most often utilized class decomposition techniques [115].

This thesis places an emphasis on undersampling because the demand for resources due to the reduced sample size is lower.

4.5.1 Undersampling Methods

Random Undersampling

The simplest class-balancing strategy is Random Undersampling (RUS) [119]. In order to generate a balanced dataset, instances of the majority class data S_{maj} are randomly removed [120]. In this approach, the total number of cases in S_{maj} is decreased by E and the balance of the class distribution of the dataset S is modified appropriately. The reduction continues until the minority and majority classes are distributed equally. The result can be defined as: $S' = S + S_{maj} + S_{min} - E$.

Near Miss

In 2003 [121], Mani and Zhang introduced a novel technique known as Near Miss Undersampling that uses the K-nearest neighbor (KNN) algorithm. The Near Miss algorithm computes the average distances between a given point and the nearest or farthest points of the opposite class. It attempts to balance the distribution of classes by removing samples from the majority class. One option for measuring distance is the Euclidean distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.13}$$

The average distance for a majority instance x_i and the three nearest or farthest instances of the minority class can be defined as:

$$d_i = \frac{1}{3} \sum_{k=1}^{3} d_{ik} \tag{4.14}$$

The Near Miss Algorithm has three versions:

- **Near Miss - Version 1:** This method selects majority class instances for which the average distance to the three nearest instances of the minority class is the least.
- **Near Miss - Version 2:** This method selects majority class instances for which the average distance to the three farthest instances of the minority class is the least.
- **Near Miss - Version 3:** This method selects for each minority instance, a predefined number of closest majority instances.

Cluster Centroids

Based on the k-means clustering approach, Altmcay and Ergün [122] established the idea of Cluster Centroids (CC). This method reduces the majority class by replacing a cluster of majority samples with the cluster centroid.

K-means, one of the simplest unsupervised learning algorithms, was first presented by Macqueen in 1967 and has since been employed by other academics to address a number of well-known clustering issues [123]. The method divides a given dataset into a certain number (k) of clusters. Briefly, the algorithm first randomly initializes the mean values of k clusters, called centroids. The next step is to calculate the distance between each data point and a centroid in order to assign the data point to the nearest centroid. One of the most prevalent approaches for measuring this distance is the use of the Euclidean distance in Eq. 4.13. The cluster centroids are then re-calculated by averaging all of the data points assigned in each cluster [124].

Let N_t and N_i denote the number of training samples from minority and majority classes respectively where $N_i \gg N_t$. The instances of the majority class are assigned into a number of different clusters equal to the number of the minority class samples (i.e. $k = N_t$) [125]. The most similar instances should be grouped into one cluster, whereas the dissimilar instances will be placed in other clusters[125]. This approach replaces the N_t majority samples by the N_t cluster centroids, thus reducing the population of the majority samples.

The fundamental aim of this algorithm is to balance a dataset by selecting informative majority class instances rather than randomly eliminating the majority class instances[126].

Neighborhood Cleaning Rule

The Neighborhood Cleaning Rule (NCR) is an undersampling technique introduced by Laurikkala [127]. Its key benefit is that it prioritizes data cleaning over reduction while taking into account the data's quality. NCR combines the Wilson's Edited Nearest Neighbor Rule (ENN) [128] and the 3NN rule to remove misclassified instances. Two sets of misclassified samples are formed and removed from the original dataset. Specifically,

- A_1 is created by the implementation of ENN. ENN algorithm identifies and removes instances from the majority class that are likely to be misclassified as the minority class. The algorithm works by considering the 3 nearest neighbors of each majority instance. If the class label of an majority instance is different from the class of at least two of its 3 nearest neighbors, then the instance is considered to be misclassified and is removed from the majority class [129].
- A_2 is created by considering the three nearest neighbors (3-NN) of each minority class instance. If the 3 nearest neighbors of a minority class instance are incorrectly classified, i.e., they belong to the majority class, then these 3 nearest neighbors are removed from the majority class [129].

Multi-class imbalance management

Dealing with multi-class imbalanced problems is more difficult than dealing with binary ones. The roles of the classes are clearly defined in binary problems: one class is the majority, while the other is the minority. On the other hand, when dealing with many classes, one class may simultaneously be the majority compared to some, the minority compared to others, and even in a relative balance with other classes. As a result, constructing balancing algorithms that take into account these varied roles is difficult[120].

The majority of the under-sampling methods that were previously analyzed for binary data are not directly applicable to multiclass classification [130]. Current approaches dealing with multiclass imbalances are primarily based on modifying techniques for binary preprocessing. According to this, the most popular approaches rely on the decomposition of multi-class problems into binary ones with the One-Versus-One (OVO) [131] or the One-Versus-Rest (OVR) technique [132].

OVR is a framework for dividing multi-class data into many binary sub-problems [133]. More specifically, from the original dataset D, the number of classes c, is used to generate a series of individual datasets, D_1, D_2,..., and D_c. Each dataset D_i contains all instances of D, but the class label y is replaced by δ_{yi} where δ is the well known kronecker delta. A binary classifier C_i is trained on each dataset D_i. In this scenario, a number of c classification tasks are generated for the dataset. For a new instance x, the class is normally assigned by the component classifier with the highest confidence based on Eq. 4.15 [130].

$$\text{argmax}_{i\epsilon\{1,2,\ldots,c\}}\, C_i\,(x) \tag{4.15}$$

In contrast to the OVR, the OVO aims to split the multi-class problem into multiple binary sub-problems, with each sub-problem focusing on a pair of classes [133]. A dataset D_{ij} that includes instances of classes i and j is created for each pair of classes and it is used to train abcorresponding inary classifier C_{ij}. The final class determination of x is made by majority voting as in Eq. 4.16 [130].

$$\text{argmax}_{i\epsilon\{1,2,\ldots,c\}} \sum_{j=1}^{c} C_{ij}\,(x) \tag{4.16}$$

Notably, this technique generates a quadratic number of classification tasks $(c\,(c-1)\,/2)$ [130]. While Near Miss is originally designed for binary classification, it can be extended to multi-class classification by applying it separately to each pair of classes. For example, we can use it to under-sample the samples of

class 1 that are closest to class 2, then use it again to under-sample the samples of class 2 that are closest to class 3.

Table 4.5 shows how each undersampling method is extended to handle multiclass classification. The imbalanced-learn library in Python provides an implementation of all the under-sampling techniques mentioned above.

Table 4.5: Extension of undersampling methods into multiclass classification problems.

Undersampling Method	Extented Technique
Random Undersampling	OVR
Cluster Centroids	OVR
Near Miss	OVO
Neighborhood Cleaning Rule	OVO

Weighted Random Sampler

Most of the time, in deep learning projects, it is difficult to pass the entire dataset into the neural network at once. Therefore, the dataset is divided into a number of batches. Experiments using imbalanced datasets may produce batches with unequal representation of classes in favor of the majority class.

However, Pytorch can produce a training batch with equal representation of each class with the *Weighted Random Sampler* technique. Weighted random sampling is a strategy for selecting items at random from a population in which certain items have a larger likelihood of being chosen than others. This is accomplished by assigning a weight to each item in the population, with the weight representing the likelihood of the item being chosen. The weights are used to generate a probability distribution over the population, and a random sample is selected from this distribution.

The key idea introduced by Pytorch is the development of a multinomial distribution, a common distribution for characterizing categorical variables, to represent the observations. Each observation is assigned a probability of being sampled.

For simplicity, the implementation of the weighted random sampler will be developed to deal with a binary classification problem. Assuming that there are N observations belonging to one of the classes c_0 and c_1, the N can be defined as:

$$N = N_0 + N_1, \quad N_0 \geqslant N_1 \tag{4.17}$$

,where

- N_0 equals the number of samples that belong to class c_0
- N_1 equals the number of samples that belong to class c_1

Firstly, the weights for each class need to be determined. It is common, for each class to be assigned with the inverse of its empirical priors. In addition, the weights can be defined as:

$$W_0 = \frac{N}{N_0} \tag{4.18}$$

$$W_1 = \frac{N}{N_1} \tag{4.19}$$

Then, a multinomial distribution with specified parameters is generated. Each parameter specifies the likelihood of drawing a given observation. The probabilities are obtained by simply normalising the weight vector:

$$p_0 = \frac{W_0}{N_0 W_0 + N_1 W_1} \tag{4.20}$$

$$p_1 = \frac{W_1}{N_0 W_0 + N_1 W_1} \tag{4.21}$$

m Points sampled from a multinomial distribution on the whole set of points $X : Multinomial(X, [p_0, ..., p_0, p_1, .., p_1])$ are generated and defined as: $X^{(1)}, X^{(2)}, ..., X^{(m)}$.
A random variable that indicates whether a point belongs to class c_0 can be defined as follows:

$$Y_i^{(0)} = \begin{cases} 1, & X^i \ from \ class \ c_0 \\ 0, & \text{otherwise.} \end{cases} \tag{4.22}$$

The expected value of such a random variable is:

$$E\left[Y_i^{(0)}\right] = p\left(Y_i^{(0)} = 1\right) = p\left(X^i \ \epsilon \ c_0\right) = N_0 p_0 \tag{4.23}$$

The random variable corresponding to the total number of points sampled from class c_0:

$$Y^{(0)} = \sum_{i=1}^{m} Y_i^{(0)} \tag{4.24}$$

As a result, the expected number of elements sampled from class c_0 after m iterations is calculated such as:

$$E[Y^{(0)}] = \sum_{i=1}^{m} E[Y_i^{(0)}] = mN_0 p_0 = mN_0 \left(\frac{W_0}{N_0 W_0 + N_1 W_1}\right) \tag{4.25}$$

$$E[Y^{(0)}] = mN_0 \left(\frac{\frac{N}{N_0}}{2N}\right) = \frac{m}{2} \tag{4.26}$$

In other words, the minority class will account for exactly half of the sampled observations despite the overall class imbalance.

4.5.2 Cost Sensitive Learning

In addition to sampling strategies, cost-sensitive methods are considered important solutions to class imbalance issues. Most machine learning algorithms struggle with imbalanced classes of data. However, the training algorithms can be adjusted in order to account for the imbalanced class distribution by assigning different weights to the majority and minority classes [134]. The overall goal is to penalize minority class miss-classification by assigning a larger class weight to the minority class while decreasing the weight for the majority class [135].

Cost-Sensitive SVM

The SVM algorithm searches for a hyperplane to divide the classes [134]. The hyperplane is defined by a margin that optimizes the distance between the decision boundary and the nearest examples from each of the two classes.

Typically, the classes cannot be linearly separated easily. Therefore, the margin is widened to allow some points to fall on the incorrect side of the decision boundary. This softening of the margin is controlled by a regularization hyperparameter known as C. In general, C manages the trade-off between enhancing the separation margin between classes and reducing the number of instances that are incorrectly classified [134]. Given n data points for training, the soft margin optimization task (see Eq. 4.27) can be written as:

$$\begin{aligned} &\underset{\mathbf{w},\xi,\mathbf{b}}{min} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right\} \\ &\text{subject to} \underset{\mathbf{i = 1,\ldots,n}}{\forall} \xi_i \underset{\mathbf{0}}{\geqslant} y_i \left(\mathbf{w} \cdot \Phi \left(\mathbf{x_i} + b \right) \right) \geq 1 - \xi_i \end{aligned} \tag{4.27}$$

where the penalty term $\sum_{i=1}^{n} \xi_i$ shows how many training instances may be on the incorrect side of the decision boundary throughout the classifier learning process [134].

When fitting the model, specifically when determining the decision boundary, the C parameter is employed as a penalty. The default weighting for each class is the same, so the softness of the margin is symmetrical. Given that the majority class has a larger number of examples than the minority class, the decision boundary will favor the majority class focusing on a better trade-off between classification error and margin maximization[134].

Adding a weight to the C value based on the relevance of each class is perhaps the most straightforward and often-used adjustment to SVM for imbalanced classification. More specifically, the penalty term C used to calculate the margin for fitting the SVM model is unique to each sample in the training dataset [136]. The C_i value of an example i can be computed by weighting the global C proportionally to the class distribution such as in Eq. 4.28.

$$C_i = weight_i \cdot C \tag{4.28}$$

For the minority class, a larger weight can be used, allowing the margin to be softer, whereas a smaller weigh can be used for the majority class, causing the margin to be harder and preventing misclassified examples. The objective function 4.27 is modified as in 4.29:

$$\begin{aligned} &\underset{\mathbf{w},\xi,\mathbf{b}}{min} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^{n} C_i \cdot \xi_i \right\} \\ &\text{subject to} \underset{\mathbf{i = 1,\ldots,n}}{\forall} \xi_i \underset{\mathbf{0}}{\geqslant} \underset{\mathbf{0}}{\forall} C_i \geq \underset{\mathbf{0}}{\forall} y_i \left(\mathbf{w} \cdot \Phi \left(\mathbf{x_i} + b \right) \right) \geq 1 - C_i \xi_i \end{aligned} \tag{4.29}$$

Cost-Sensitive Decision Tree

The class distribution is altered by the instance-weighting approach in such a way that the resulting tree is advantageous to the class with the highest weight and is less prone to make costly errors.

In general, a tree is built by dividing the training dataset, with instances moving through its decision points until they reach its leaves and get a class label. A greedy decision is made at each point to divide the data in a way that produces the purest (least mixed) groups of samples. A purity measure is calculated by estimating the likelihood of a split misclassifying an example of a given class. However, it is possible to update the splitting criterion to weigh the relative importance of each class in addition to the purity of the split.

Assume that N is the total number of examples from the training set, and $\mathrm{N_j}$ is the number of examples from class j. Additionally, $N(t)$ represents the number of instances and $\mathrm{N_j(t)}$ represents the number of class j instances in node t of a decision tree. The ratio of the total number of instances in class j to the total number of instances in this node shows how likely it is that an instance in node t belongs to class j (See Eq. 4.30) [137].

$$p\left(j \mid t\right) = \frac{N_j(t)}{\sum_i N_i(t)} \tag{4.30}$$

The cost-sensitive tree is based on modifying an instance's weight in a proportional manner to the cost of misclassifying the class to which the instance belongs while maintaining a constant value of N for the total weight of all training instances. Given that C_j is the cost of missclassifying an instance of class j, the weight of a class j instance can be calculated as follows:

$$w(j) = C(j)\frac{N}{\sum_i C(i)N_i} \tag{4.31}$$

such that the sum of all instance weights is $\sum_j w(j)N_j = N$

Similar to $p(j|t)$, $p_{\mathrm{w}}(j|t)$ is defined as the ratio of the total weight of class j instances to the total weight of node t:

$$p_w(j|t) = \frac{w(j)N_j(t)}{\sum_i w(i)N_i(t)} \tag{4.32}$$

In order to improve (lessen) a node's purity score and make it appear more sorted, a small weight could be given to the majority class and a larger weight to the minority class. With this adjustment, the normal tree induction method which aims to eliminate errors regardless of cost is effectively changed into one that prioritizes errors with a high weight.

Cost-Sensitive Random Forest

Cost-sensitive learning is another way to improve Random Forest for highly imbalanced data. Since the RF classifier tends to favor the majority class, the misclassification of the minority class could be penalised more severely. Each class is assigned a weight, with the minority class receiving a higher weight.

The RF algorithm incorporates the class weights twice. Class weights are used to adjust the Gini or Entropy criterion for partitioning data in the tree induction process [138]. The calculation of impurity could be skewed in such a way that it favors a mixture that is more representative of the minority class, leading to the possibility of some false positives of the majority class.

Class weights are considered once more at each tree's leaf nodes. The class prediction of each terminal node is determined by "weighted majority vote"; i.e., the weighted vote of a class equals the class's weight multiplied by the number of cases for that class at the terminal node. The final class prediction for RF is then calculated by averaging the weighted vote from each individual tree, where the weights are the average weights in the terminal nodes [138].

Calculation of class weights

The weights are calculated using the inverse proportion of class frequencies, where heavier weights are assigned to underrepresented classes in order to give them more prominence during training. More specifically, the weight for a class i is calculated based on Eq. 4.33.

$$w_i = n_{samples}/(n_{classes} \cdot n_{samples_i}) \tag{4.33}$$

where,

- w_i is the weight for class i
- $n_{samples}$ is the total number of samples of the training set
- $n_{classes}$ is the total number of unique classes
- $n_{samples_i}$ is the total number of samples in the class i

For example, if we utilize the Patient-Wise Splitting 1 which has 29917 samples in total and 3306 samples in weak pain class, the class weight of weak pain can be computed as 29917 / $(3 \cdot 3306)$ = 0.40350948. Tables 4.6, and 4.7 show the weights calculated for Patient-Wise Splitting 1 and Patient-Wise Splitting 2 respectively.

Table 4.6: Class weights assigned to Patient-Wise Splitting 1

	No pain	Weak Pain	Strong Pain
Number of samples	26764	3452	2274
Class weight	0.4046	3.1373	4.7625

Table 4.7: Class weights assigned to Patient-Wise Splitting 2

	No pain	Weak Pain	Strong Pain
Number of samples	24714	3306	1897
Class weight	0.4035	3.0164	5.2568

4.6 Hyperparameter Optimization

The term *model* is referring to a combination of a specific Machine Learning algorithm and its *parameterization*, i.e. the particular values that have been selected for the algorithm's tunable or definable parameters. These parameters may be structural, such as the number of hidden layers in a neural network or the number of estimators in the Random Forest algorithm, or algorithmic, such as the minibatch size or learning rate, which regulate the learning process. Hyper-parameters control the learning process and need to be defined as arguments before the training phase begins.

Default hyperparameters are typically provided in software packages. If they are set up right, they might give good results for a certain problem. However, the performance is often improved over the default by "tuning" the hyperparameters or finding their optimal values. When performing a search, it is necessary to define a range of possible values for each hyperparameter. The size of the range can significantly affect the performance of certain optimization techniques.

4.6.1 Grid Search

Grid search is the most fundamental method for automating hyperparameter optimization [139]. Initially, the user specifies a grid with distinct values for each hyperparameter to be tested. Then, the given algorithm is evaluated with all of the possible combinations of those values. The optimal configuration is returned as the one that achieves the highest score in the evaluation metric on the validation set. Grid search can be computationally expensive for models with many hyperparameters or a large number of possible values for each hyperparameter.

4.6.2 Random Search

Instead of specifying a set of values for each hyperparameter, the user provides a range to search values from [139]. The values for each hyperparameter are randomly sampled from the given hyperparameter space, such as using the uniform distribution, rather than trying all possible combinations. The number of random combinations can also be specified. In comparison with Grid Search, in Random Grid Search, finding the optimal hyperparameter setup needs less time and resources. However, this method is less likely to find the optimal solution.

4.6.3 Optuna

Optuna is a software framework for automating the hyperparameter optimization process [140]. The framework includes an API (application programming interface) that allows users to dynamically construct their own search space for parameters, as well as to implement pruning and sampling strategies [141]. In Optuna, trials are used to keep track of multiple attempts to optimize an objective function, with each trial storing information about the hyperparameters tried and the corresponding results.

In order to find the best hyperparameters, Optuna in the default status, employs a Bayesian optimization algorithm and a tree-structured Parzen estimator as a sampler [140]. Samplers continuously reduce the search space by using information of previously records, creating an optimal search space that obtains parameters that result in better objective values.

Pruning mechanism terminates the unpromising trials during hyperparameter optimization. It monitors the learning curves of each trial in order to evaluate their progress compared to previously achieved results and terminates training in early stages for specific hyperparameter combinations that aren't producing good results. Optuna's pruning mechanism is based on an asynchronous variant of the Successive Halving Algorithm (SHA). It works by separating the set of trials into many rounds, with only a subset of the trials advancing to the next round based on their performance. In the first round, all trials are evaluated using the validation set and the top N percent of trials with the best performance are chosen to advance to the next round. In succeeding rounds, the same procedure is performed with the remaining trials, while the number of trials in each round is gradually reduced by one-half. This process is repeated until only one trial remains,

which is selected as the best combination of hyperparameters.
Both pruning and sampling significantly reduce the time for the hyperparameter optimization procedure.

4.7 Model Evaluation

In an imbalanced dataset, many machine learning algorithms are unable to correctly predict examples from the minority classes. On the contrary, they can calculate high overall accuracy on the whole dataset, as the model is likely to successfully classify most of the instances belonging to the majority class. However, since the minority classes are frequently the classes of interest, the accuracy metric is not representative. As a result, more informative metrics, such as f-measure, precision, and recall, are recommended to estimate the models' performance.

The usage of a confusion matrix is a popular technique for assessing a classifier's effectiveness [142]. The confusion matrix for a multi-class classification has a dimension N × N where N is the number of different class labels C_0, C_1, . . . , C_N. We will conduct an analysis focusing on the class of interest i. Each prediction of a multi-class classifier can fall under one of the following four categories:

- True positive (TP): An instance where the model correctly predicts the class of interest.
- False positive (FP): An instance where the model predicts the class of interest, but the actual class is different.
- True negative (TN): An instance where the model correctly predicts a class that is not the class of interest.
- False negative(FN): An instance where the model predicts a class that is not the class of interest, but the actual class is the class of interest.

For the required computations, we will use the labeling of the Confusion Matrix in Figure 4.18 focusing on the class C_2 [115]. In particular, we consider True Positive (TP) only the correctly classified instances for the class of interest, whereas False Positive (FP) and False Negative (FN) are the miss-classified elements on the column and the row of the class respectively. True Negative (TN) are all the other tiles, as shown in 4.18. For each class, the the labels of the Confusion Matrix tiles are adjusted accordingly.

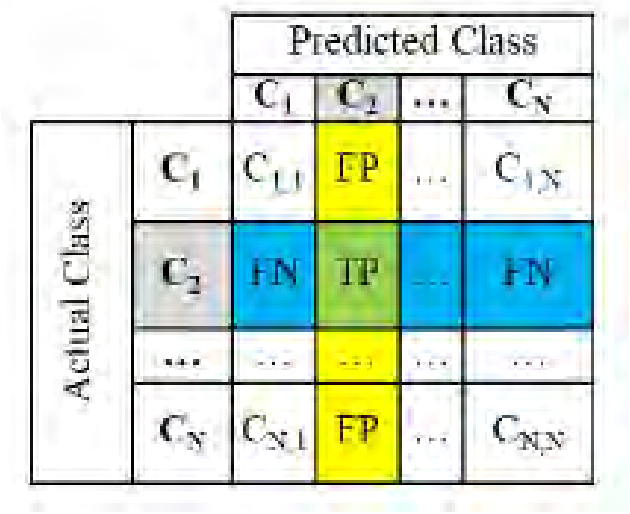


Figure 4.18: Multiclass classification problem confusion matrix, [12]

Firstly, the Precision ($\text{Precision}_\text{i}$) and the Recall ($\text{Recall}_\text{i}$) for a class i can be computed based on the confusion matrix as [115], [12], [143]:

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$
$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

where,

- TP_i (True Positive) is the i-th diagonal element,
- FP_i (False Positive) is the sum of off-diagonal elements of the i-th row and
- FN_i (False Negative) is the sum of the off-diagonal elements of the i-th column.

Precision shows how often an instance that was predicted as a particular class is actually that class, while recall measures how often a positive instance of that class was predicted as that class by the classifier. In other words, recall indicates the model's capacity to identify the instances of a particular class, whereas precision illustrates how much we can rely on the model when it forecasts an instance as belonging to that class [115].

The objective in imbalanced datasets is to increase recall without sacrificing precision. This objective, however, frequently conflicts because it is common practice to increase the TP for the minority class while concurrently increasing the number of FP, which lowers precision. Consequently, the F_1 score was introduced as the harmonic mean of precision and recall to balance the two metrics. Two different types of F_1 scores are frequently computed for multiclass classification tasks: the micro-averaged F_1 score and the macro-averaged F_1 score. Briefly, the micro-averaged F_1 score determines the overall F_1 score by combining per-sample classifications across classes while the macro-averaged F_1 score computes a straightforward average of the f1 scores across classes [144]. We selected the F_1 score with a macro average as the assessment metric for model selection since it calculates the f-score for each label and provides their unweighted mean, allowing each class to have the same weight in the average regardless of the number of instances [12]. In this way, it does not prioritize the majority class and is more resistant to class imbalance.

Macro averaged F_1 score is obtained by calculating the Macro averaged Precision and Macro averaged Recall. Macro Averaged Precision and Macro Averaged Recall are simply computed as the average precision and the average recall for all class:

$$MacroAveragePrecision = \frac{1}{N}\sum_{i=1}^{N} Precision_i \tag{4.34}$$

$$MacroAverageRecall = \frac{1}{N}\sum_{i=1}^{N} Recall_i \tag{4.35}$$

Finally, Macro averaged F_1 score is the harmonic mean of Macro-Precision and Macro-Recall:

$$MacroF1Score = 2 \cdot \frac{MacroAveragePrecision \cdot MacroAverageRecall}{MacroAveragePrecision + MacroAverageRecall} \tag{4.36}$$

4.8 Tools

In this research, we make use of several powerful tools and libraries to analyze and model the data. The primary programming language used is Python, and we utilize a number of popular libraries such as numpy and pandas for data manipulation and preprocessing. We also make use of visualization libraries such as matplotlib and seaborn to create clear and informative plots and figures to aid in our analysis and interpretation of results. In addition, we employ several machine learning libraries for building and evaluating our models such as :

1. **Scikit-learn**: Scikit-learn is a Python library for machine learning that provides algorithms for classification, regression, clustering, and dimensionality reduction. It is user-friendly, efficient, and flexible, built on top of NumPy, SciPy, and Matplotlib. It has a variety of tools for model optimization such as cross-validation and hyperparameter tuning, and various metric functions for model evaluation.
2. **PyTorch**: PyTorch is a Python deep learning library providing a flexible and high-performance environment for the development and training of deep learning models. It is widely used in natural language processing and computer vision, and comes with a number of built-in libraries for data loading, image preprocessing, and model evaluation.
3. **Optuna**: Optuna is a Python library for hyperparameter optimization. It aims to simplify and speed up the process of fine-tuning the parameters of machine learning models. Optuna employs different optimization techniques to search the optimal set of hyperparameters for a given model. Additionally, it offers features such as early stopping, pruning of ineffective trials, and the ability to store and retrieve trials from a database, which allow users to monitor the optimization process.
4. **Hypopt**: Hypopt is another Python library to perform hyperparameter optimization using a validation set.
5. **Imbalanced-learn**: Imbalanced-learn is a Python library for handling imbalanced datasets in machine learning. It addresses the imbalance issue using a variety of strategies, including oversampling the minority classes and undersampling the majority classes. A variety of pre-processing algorithms and metrics for evaluating the performance of machine learning models on imbalanced datasets are also included.
6. **Albumentations**: Albumentations is a Python library for image augmentation. The library includes a variety of pre-defined augmentation techniques, including flipping, rotating, cropping, scaling, colour jittering, as well as the option to build unique transformations. The library is written in Python and represents images as numpy arrays, making it ideal for usage with deep learning frameworks like TensorFlow and PyTorch. It is also allowing users to simply layer numerous augmentations together to create diverse data augmentations.
7. **Scikit-image**: Scikit-image is a Python library for image processing based on NumPy and SciPy. It provides a wide range of algorithms for image processing tasks such as filtering, morphological operations, feature extraction, and segmentation.
8. **SciPy**: SciPy is a Python library for scientific and technical computing. It extends the NumPy library with additional numerical calculation capability such as optimization methods, signal processing, and image processing.

The experiments were performed on Google Colab utilizing both GPU and CPU capabilities. Overall, these tools and libraries provide a robust and flexible framework for our research, allowing us to efficiently work with the data.

5 Results

In this study, we evaluated the performance of both simple classifiers and neural networks in order to predict three intensities of facial pain. We experimented with different techniques such as undersampling, cost-sensitive learning, transfer learning, and data augmentation to improve the performance of our models. Our results provide insights into the strengths and limitations of each model.

5.1 Results of Traditional Classifiers

We implemented three different types of machine learning models for classification: Support Vector Machines (SVMs), Decision Trees (DTs), and Random Forests (RFs). The framework for building a machine learning model to address imbalanced multiclass problems involves the following steps:

1. Define the dataset splitting strategy
2. Select the machine learning algorithm
3. Specify the algorithm's hyperparameter space
4. Determine the method to address imbalance
5. Determine the hyperparameter search method
6. Choose the optimization metric

To begin with, we propose two different dataset splitting strategies that are based on patient-wise splitting. Specifically, we experiment with Patient - Wise Splitting 1 and Patient - Wise Splitting 2. It is important to note that both methods produce a test set that is not used during the construction and the training of the model but only for its final evaluation with the aim of estimating the degree of generalization. Then, we have to decide how to address the imbalance problem. In our work, we explore two different techniques: undersampling and cost-sensitive learning. The undersampling methods we employ are: Random Undersampling, Near Miss, Neighborhood Cleaning Rule, and Cluster Centroids. Furthermore, to improve the performance of a model, we use techniques such as finetuning, in which we search the optimal set of hyperparameters for the model. Also we make use of model selection techniques, in which we compare the performance of different models on the validation data and select the optimal one for further evaluation. The metric we chose to examine for this procedure is f1 macro as it gives a balanced view of the model's performance across all classes. After selecting the best model, we evaluate its performance on the test set. This gives us an estimate of the model's performance on unseen data and allows us to make an informed decision about whether to use the model in a real-world scenario.

In SVMs, we used the Grid Search method to explore a range of values in order to optimize the following parameters:

- $Kernel$: This parameter specifies the kernel function to be utilized in the SVM model.
- C : This parameter governs the trade-off between maximisation of margin and minimization of classification error. A lower C value results in a wider margin but may allow more misclassifications, whereas a higher C value results in a smaller margin but fewer misclassifications.
- $gamma$: This parameter, when combined with the RBF kernel, determines the width of the radial basis function. A lower gamma value produces a wider radial basis function, while a higher gamma value produces a narrower radial basis function.
- $decision\_function\_shape$: This parameter is used to configure the decision function. In our implementation, it is set to 'ovr' (one-vs-rest).

The search space for each of the SVM's hyperparameters is defined in Table 5.1. The svm.SVC (Support Vector Classification) class of the scikit-learn library is used to train and make predictions with a SVM model. In addition, GridSearch class of the Hypopt library is used to implement Grid Search hyper-parameter optimization using a validation set for the Patient-Wise Splitting 1, while GridSearchCV class of scikit-learn library performs Grid Search using Stratified-group k-fold cross-validation for the Patient-Wise Splitting 2.

Table 5.1: Hyperparameter space of SVM

Parameter	Search Space
Kernel	[rbf, linear]
C	[0.001, 1, 10, 100]
gamma	[0.001, 0.01, 0.1, 1, 10, 100]

RF and DT have similar parameters that can be optimized to improve their performance, some of the most important ones include:

- max_depth : This parameter determines the maximum depth of a tree. A higher value produces more complicated trees, while a lower value produces simpler trees. Although deeper trees have the potential to capture more nuanced relationships in the data, they also increase the risk of the model overfitting.
- $min\_samples\_split$: This parameter sets the minimum number of samples that an internal node must have before it can be split. By increasing this value, the tree becomes more conservative and the risk of overfitting is reduced.
- $min\_samples\_leaf$: This parameter specifies the minimum number of samples that must be present at a leaf node. If the number of samples in a leaf following a split is less than $min\_samples\_leaf$, no further splits will be performed.
- $max\_features$: This parameter specifies the number of features to evaluate when determining the optimum split.
- $criterion$: Both algorithms employ either gini or entropy as a splitting criterion.

In addition, for RF the $n\_estimators$ can be optimized which determines the total number of Decision trees in the forest. A larger number of trees produces a more robust model but increases the computing cost. Correspondingly, a parameter that can be additionally tuned for the DT is max_leaf_nodes. It specifies the maximum number of leaf nodes in a tree. We can ensure that the tree does not get excessively complex and that it generalises effectively to new data by limiting the number of leaf nodes. If the max_leaf_nodes

parameter is not specified, the tree can keep expanding indefinitely, which may result in overfitting. However, if this parameter is set to a low value, the tree may underfit and fail to capture enough data-derived information.

Table 5.2: Hyperparameter space of Random Forest

Parameter	Search Space
n_estimators	Integers within the range of 10 to 500 and the single value of 1000
max_depth	Integers within the range of 10 to 100 with a step size of 10 which starts from 10 and ends at 90.
min_samples_split	Integers within the range of 2 to 12 with a step size of 2 which starts from 2 and ends at 10. And also additional values 20 and 50.
min_samples_leaf	Integers within the range of 1 to 5 with a step size of 1 which starts from 1 and ends at 4.
max_features	Integers within the range of 1 to 8 with a step size of 1 which starts from 1 and ends at 7. And also additional values of 'auto' and 'sqrt'.
criterion	gini, entropy

Table 5.3: Hyperparameter space of Decision Trees

Parameter	Search Space
max_depth	Integers within the range of 1 to 9.
min_samples_leaf	Integers within the range of 1 to 9.
max_features	Integers within the range of 1 to 9.
criterion	gini, entropy
n_estimators	Integers within the range of 10 to 100 with a step size of 10 which starts from 10 and ends at 90.

Tables 5.2 and 5.3 define the search spaces for the hyperparameters of RF and DT respectively. From the scikit-learn library, The RandomForestClassifier class is used to train a Random Forest model and the DecisionTreeClassifier class to train a Decision Tree model. Optuna is used to find the best set of hyperparameters using the training and the validation sets of Patient-Wise Splitting 1. Furthermore, RandomizedSearchCV class of scikit-learn performed Randomized Search using Stratified-group k-fold cross-validation for the Patient-Wise Splitting 2.

The performance of the classifiers was evaluated using several metrics, including precision and recall for each class and F1 macro score. The following tables present the results for each of the three models, SVM (Tables 5.4 - 5.8), Random Forest (Tables 5.9 - 5.13), and Decision Tree (5.14 - 5.18), with each of the five features vectors: maximum angles, minimum angles, areas, distances, and HOG features. Experiments utilizing HOG features could not be completed due to excessive resource requirements.

Additionally, Tables 5.19 and 5.20 present the results of leave-one-subject-out evaluation for the top-performing SVM, Decision Tree, and Random Forest models for each characteristic. These models were selected based on their performance on the test sets. The chosen models were trained for 25 iterations, where in each iteration 24 patients are used in the training set and one patient is left out for testing.

Support Vector Machines	Maximum Angles								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	GridSearch	x	x	$kernel = linear$ $C = 100$	$\begin{bmatrix} 7470 & 273 & 204 \\ 932 & 38 & 95 \\ 428 & 34 & 154 \end{bmatrix}$	0 1 2	0.85 0.11 0.34	0.94 0.04 0.25	0.41
	GridSearch	x	✓	$kernel = rbf$ $C = 10$ $gamma = 0.001$	$\begin{bmatrix} 6307 & 1006 & 634 \\ 754 & 131 & 180 \\ 276 & 76 & 264 \end{bmatrix}$	0 1 2	0.86 0.11 0.24	0.79 0.12 0.43	0.42
	GridSearch	NearMiss 1	x	$kernel = rbf$ $C = 1$ $gamma = 0.001$	$\begin{bmatrix} 4741 & 2522 & 204 \\ 561 & 80 & 424 \\ 156 & 63 & 397 \end{bmatrix}$	0 1 2	0.87 0.60 0.26	0.60 0.08 0.64	0.37
	GridSearch	NearMiss 3	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	$\begin{bmatrix} 6528 & 733 & 686 \\ 961 & 52 & 52 \\ 412 & 51 & 153 \end{bmatrix}$	0 1 2	0.83 0.06 0.17	0.82 0.05 0.25	0.36
	GridSearch	Neighbourhood Cleaning Rule	x	$kernel = linear$ $C = 10$	$\begin{bmatrix} 7444 & 284 & 219 \\ 930 & 38 & 97 \\ 412 & 44 & 160 \end{bmatrix}$	0 1 2	0.85 0.10 0.34	0.94 0.04 0.26	0.41
	GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	$\begin{bmatrix} 6395 & 1033 & 519 \\ 754 & 159 & 152 \\ 313 & 108 & 195 \end{bmatrix}$	0 1 2	0.86 0.12 0.23	0.80 0.15 0.32	0.41
	GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 10$ $gama = 0.001$	$\begin{bmatrix} 6021 & 1367 & 559 \\ 766 & 84 & 215 \\ 355 & 54 & 207 \end{bmatrix}$	0 1 2	0.84 0.06 0.21	0.76 0.08 0.34	0.37
	GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 10$ $gamma = 0.001$	$\begin{bmatrix} 6021 & 1367 & 559 \\ 766 & 84 & 215 \\ 355 & 54 & 207 \end{bmatrix}$	0 1 2	0.84 0.06 0.21	0.76 0.08 0.34	0.37
	Patient-Wise Splitting 2								
	GridSearch	x	x	$kernel = rbf$ $C = 100$ $gamma = 0.01$	$\begin{bmatrix} 10611 & 1590 & 1064 \\ 797 & 362 & 649 \\ 523 & 58 & 254 \end{bmatrix}$	0 1 2	0.89 0.18 0.13	0.80 0.20 0.30	0.40
	GridSearch	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.01$	$\begin{bmatrix} 9898 & 2319 & 1048 \\ 242 & 684 & 882 \\ 330 & 186 & 319 \end{bmatrix}$	0 1 2	0.95 0.21 0.14	0.75 0.38 0.38	0.44
	GridSearch	NearMiss 1	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	$\begin{bmatrix} 5393 & 6180 & 1692 \\ 196 & 564 & 1048 \\ 269 & 123 & 443 \end{bmatrix}$	0 1 2	0.92 0.08 0.14	0.41 0.31 0.53	0.30
	GridSearch	Near Miss 2	x	$kernel = rbf$ $C = 0.001$ $gamma = 1$	$\begin{bmatrix} 13265 & 0 & 0 \\ 1808 & 0 & 0 \\ 835 & 0 & 0 \end{bmatrix}$	0 1 2	0.83 0.00 0.00	1.00 0.00 0.00	0.30
	GridSearch	NearMiss 3	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	$\begin{bmatrix} 9255 & 2109 & 1901 \\ 785 & 256 & 767 \\ 446 & 91 & 298 \end{bmatrix}$	0 1 2	0.88 0.10 0.10	0.70 0.14 0.36	0.35
	GridSearch	Neighbourhood Cleaning Rule	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	$\begin{bmatrix} 12283 & 543 & 439 \\ 901 & 356 & 551 \\ 582 & 23 & 230 \end{bmatrix}$	0 1 2	0.89 0.39 0.19	0.93 0.20 0.28	0.46
	GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	$\begin{bmatrix} 10314 & 1654 & 1297 \\ 397 & 605 & 806 \\ 433 & 146 & 256 \end{bmatrix}$	0 1 2	0.93 0.25 0.11	0.78 0.33 0.31	0.43
	GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 100$ $gamma = 0.01$	$\begin{bmatrix} 10583 & 1396 & 1286 \\ 678 & 496 & 634 \\ 480 & 95 & 260 \end{bmatrix}$	0 1 2	0.90 0.25 0.12	0.80 0.27 0.31	0.43

Table 5.4: Experiments with SVM and Maximum Angles

Support Vector Machines	Minimum Angles									
	Patient-Wise Splitting 1									
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix			Class	Precision	Recall	F1 macro
	GridSearch	x	x	*kernel* = *rbf* $C = 1000$ *gamma* = 0.01	7171 462 314 1001 10 54 460 26 130			0 1 2	0.83 0.02 0.26	0.90 0.01 0.21	0.37
	GridSearch	x	✓	*kernel* = *rbf* $C = 1000$ *gamma* = 0.01	7519 231 197 1011 8 46 464 20 132			0 1 2	0.84 0.03 0.35	0.95 0.01 0.21	0.39
	GridSearch	NearMiss 1	x	*kernel* = *rbf* $C = 1$ *gamma* = 0.001	5239 1947 761 441 287 337 153 13 450			0 1 2	0.90 0.13 0.29	0.66 0.27 0.73	0.45
	GridSearch	NearMiss 2	x	*kernel* = *rbf* $C = 10$ *gamma* = 0.1	5180 1668 1099 473 274 318 368 5 243			0 1 2	0.86 0.14 0.15	0.65 0.26 0.39	0.38
	GridSearch	Near Miss 3	x	*kernel* = *rbf* $C = 100$ *gamma* = 0.01	7367 118 462 950 38 77 431 20 165			0 1 2	0.84 0.22 0.23	0.93 0.04 0.27	0.40
	GridSearch	Neighbourhood Cleaning Rule	x	*kernel* = *linear* $C = 1$	7580 163 204 1017 19 29 460 12 144			0 1 2	0.84 0.10 0.38	0.95 0.02 0.23	0.40
	GridSearch	Random UnderSampler	x	*kernel* = *rbf* $C = 0.001$ *gamma* = 0.01	6574 591 782 720 67 278 345 14 257			0 1 2	0.86 0.10 0.20	0.83 0.06 0.42	0.40
	Patient-Wise Splitting 2									
	GridSearch	x	x	*kernel* = *rbf* $C = 1$ *gamma* = 0.01	12267 753 245 1246 300 262 552 79 204			0 1 2	0.87 0.27 0.29	0.92 0.17 0.24	0.46
	GridSearch	x	✓	*kernel* = *linear* $C = 10$	7466 4789 1010 210 900 698 107 172 556			0 1 2	0.96 0.15 0.25	0.56 0.50 0.67	0.43
	GridSearch	NearMiss 1	x	*kernel* = *rbf* $C = 0.001$ *gamma* = 0.01	10191 1493 1581 961 46 801 308 46 481			0 1 2	0.89 0.03 0.17	0.77 0.03 0.58	0.37
	GridSearch	Near Miss 2	x	*kernel* = *linear* $C = 1$	2897 8400 1968 252 523 1033 162 168 505			0 1 2	0.87 0.06 0.14	0.22 0.29 0.60	0.23
	GridSearch	NearMiss 3	x	*kernel* = *rbf* $C = 10$ *gamma* = 0.1	9819 1632 1814 984 510 314 596 99 140			0 1 2	0.86 0.23 0.06	0.74 0.28 0.17	0.38
	GridSearch	Neighbourhood Cleaning Rule	x	*kernel* = *rbf* $C = 1$ *gamma* = 0.01	12268 733 264 1248 291 269 556 51 228			0 1 2	0.87 0.27 0.30	0.92 0.16 0.27	0.46
	GridSearch	Random UnderSampler	x	*kernel* = *rbf* $C = 1$ *gamma* = 0.01	8835 2846 1584 302 779 727 170 202 463			0 1 2	0.95 0.20 0.17	0.67 0.43 0.55	0.44
	GridSearch	Cluster Centroids	x	*kernel* = *rbf* $C = 1$ *gamma* = 0.01	9498 2646 1121 505 757 546 222 243 370			0 1 2	0.93 0.21 0.18	0.72 0.42 0.44	0.45

Table 5.5: Experiments with SVM and Minimum Angles

Support Vector Machines	Areas									
Patient-Wise Splitting 1										
Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix			Class	Precision	Recall	F1 macro
GridSearch	x	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	7712 124 111 1006 26 33 501 17 98			0 1 2	0.84 0.16 0.40	0.97 0.02 0.16	0.39
GridSearch	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.01$	7542 232 173 842 182 41 349 89 178			0 1 2	0.86 0.36 0.45	0.95 0.17 0.29	0.50
GridSearch	NearMiss 1	x	$kernel = rbf$ $C = 100$ $gamma = 0.01$	6488 1108 351 778 189 98 402 42 172			0 1 2	0.85 0.14 0.28	0.82 0.18 0.28	0.42
GridSearch	NearMiss 2	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	3424 1956 2567 139 406 520 116 92 408			0 1 2	0.93 0.17 0.12	0.43 0.38 0.66	0.34
GridSearch	Near Miss 3	x	$kernel = rbf$ $C = 100$ $gamma = 0.01$	7204 401 342 868 165 32 449 41 126			0 1 2	0.85 0.27 0.25	0.91 0.15 0.20	0.43
GridSearch	Neighbourhood Cleaning Rule	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	7703 130 114 1012 16 37 513 5 98			0 1 2	0.83 0.11 0.39	0.97 0.02 0.16	0.38
GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	7360 356 231 910 111 44 403 61 152			0 1 2	0.85 0.21 0.36	0.93 0.10 0.25	0.44
GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 1$ $gamma = 0.01$	7604 192 151 983 48 34 405 65 146			0 1 2	0.85 0.21 0.44	0.93 0.05 0.24	0.43
Patient-Wise Splitting 2										
GridSearch	x	x	$kernel = rbf$ $C = 1$ $gamma = 0.01$	11422 1087 756 874 618 316 351 60 424			0 1 2	0.90 0.35 0.28	0.86 0.34 0.51	0.53
GridSearch	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.01$	10235 1813 1217 713 730 365 298 64 473			0 1 2	0.91 0.28 0.23	0.77 0.40 0.57	0.50
GridSearch	NearMiss 1	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	8768 2501 1996 384 641 783 145 126 564			0 1 2	0.94 0.20 0.17	0.66 0.35 0.68	0.43
GridSearch	Near Miss 2	x	$kernel = linear$ $C = 10$	1533 9469 2263 230 763 815 0 239 596			0 1 2	0.87 0.07 0.16	0.12 0.42 0.71	0.20
GridSearch	NearMiss 3	x	$kernel = rbf$ $C = 100$ $gamma = 0.01$	8285 3305 1675 456 809 543 211 121 503			0 1 2	0.93 0.19 0.18	0.62 0.45 0.60	0.43
GridSearch	Neighbourhood Cleaning Rule	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	10995 1220 1050 803 586 419 301 71 463			0 1 2	0.91 0.31 0.24	0.83 0.32 0.55	0.51
GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	10165 1393 1707 803 550 455 232 79 524			0 1 2	0.91 0.27 0.20	0.77 0.30 0.63	0.47
GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 1$ $gamma = 0.01$	8465 2247 2553 525 722 561 122 140 573			0 1 2	0.93 0.23 0.16	0.64 0.40 0.69	0.43

Table 5.6: Experiments with SVM and Areas

Support Vector Machines										
Distances										
Patient-Wise Splitting 1										
Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix			Class	Precision	Recall	F1 macro
GridSearch	x	x	$kernel = linear$ $C = 10$	6996 892 59 943 84 38 426 78 112			0 1 2	0.84 0.08 0.54	0.88 0.08 0.18	0.40
GridSearch	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.001$	7117 570 260 808 200 57 368 45 203			0 1 2	0.86 0.25 0.39	0.90 0.19 0.33	0.48
GridSearch	NearMiss 1	x	$kernel = linear$ $C = 0.001$	2834 4714 399 478 372 215 212 149 255			0 1 2	0.80 0.07 0.29	0.36 0.35 0.41	0.32
GridSearch	NearMiss 2	x	$kernel = linear$ $C = 0.001$	6229 1085 633 778 47 240 345 26 245			0 1 2	0.85 0.04 0.22	0.78 0.04 0.40	0.38
GridSearch	Near Miss 3	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	7301 468 178 795 224 46 459 70 87			0 1 2	0.85 0.29 0.28	0.92 0.21 0.14	0.44
GridSearch	Neighbourhood Cleaning Rule	x	$kernel = linear$ $C = 10$	6943 934 70 923 104 38 406 91 119			0 1 2	0.84 0.09 0.52	0.87 0.10 0.19	0.41
GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	6771 823 353 534 371 160 416 79 121			0 1 2	0.88 0.29 0.19	0.85 0.35 0.20	0.46
GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 0.001$ $gamma = 0.001$	6751 755 441 533 380 152 413 86 117			0 1 2	0.88 0.31 0.16	0.85 0.36 0.19	0.46
Patient-Wise Splitting 2										
GridSearch	x	x	$kernel = rbf$ $C = 1$ $gamma = 0.001$	12253 704 308 1274 307 227 436 7 392			0 1 2	0.88 0.30 0.42	0.92 0.17 0.47	0.52
GridSearch	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.001$	8829 2323 2113 216 824 768 92 241 502			0 1 2	0.97 0.24 0.15	0.67 0.46 0.60	0.45
GridSearch	NearMiss 1	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	6708 2482 4075 171 151 1486 197 185 453			0 1 0.08	0.95 0.05 0.08	0.51 0.08 0.54	0.29
GridSearch	Near Miss 2	x	$kernel = linear$ $C = 100$	3388 3802 6075 146 223 1439 6 67 762			0 1 2	0.96 0.05 0.09	0.26 0.12 0.91	0.22
GridSearch	NearMiss 3	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	6169 2457 4639 550 79 1179 281 101 453			0 1 2	0.88 0.03 0.07	0.47 0.04 0.54	0.26
GridSearch	Random UnderSampler	x	$kernel = rbf$ $C = 10$ $gamma = 0.001$	8017 3058 2190 363 312 1133 117 193 525			0 1 2	0.94 0.09 0.14	0.60 0.17 0.63	0.36
GridSearch	Cluster Centroids	x	$kernel = rbf$ $C = 10$ $gamma = 0.01$	7952 3348 1965 439 341 1028 137 197 501			0 1 2	0.93 0.09 0.14	0.60 0.19 0.60	0.36
GridSearch	Neighborhood Cleaning Rule	x	$kernel = rbf$ $C = 100$ $gamma = 0.001$	9616 2151 1498 1007 83 718 374 129 332			0 1 2	0.87 0.04 0.13	0.72 0.05 0.40	0.34

Table 5.7: Experiments with SVM and Distances

Hog Features										
Patient-Wise Splitting 1										
Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix			Class	Precision	Recall	F1 macro
Optuna	x	x	$kernel = linear$ $C = 0.001$	6973 746 228 841 58 166 398 118 100			0 1 2	0.85 0.06 0.20	0.88 0.05 0.16	0.37
Optuna	NearMiss 1	✓	$kernel = linear$ $C = 0.01$	5258 1279 1410 530 72 463 281 65 270			0 1 2	0.87 0.05 0.13	0.66 0.07 0.44	0.33
Optuna	Neighbourhood Cleaning Rule	x	$kernel = linear$ $C = 0.001$	6896 766 285 816 40 209 396 87 133			0 1 2	0.85 0.04 0.21	0.87 0.04 0.22	0.37
Optuna	Random UnderSampler	x	$kernel = linear$ $C = 0.01$	6349 1163 435 798 77 190 380 84 152			0 1 2	0.84 0.06 0.20	0.80 0.07 0.25	0.37

Table 5.8: Experiments with SVM and Hog Features

Random Forest	Maximum Angles								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Optuna	x	x	*criterion* = *gini* *max depth* = 70 *max features* = 3 *min samples leaf* = 4 *min samples split* = 10 *n estimators* = 10	$\begin{bmatrix} 7796 & 62 & 89 \\ 897 & 19 & 149 \\ 453 & 3 & 160 \end{bmatrix}$	0	0.85	0.98	0.42
						1	0.23	0.02	
						2	0.40	0.26	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 2 *min samples leaf* = 5 *min samples split* = 16 *n estimators* = 19	$\begin{bmatrix} 7527 & 327 & 93 \\ 982 & 49 & 34 \\ 483 & 20 & 113 \end{bmatrix}$	0	0.84	0.95	0.41
						1	0.12	0.05	
						2	0.47	0.18	
	Optuna	NearMiss 1	x	*criterion* = *gini* *max depth* = 40 *max features* = 7 *min samples leaf* = 1 *min samples split* = 2 *n estimators* = 11	$\begin{bmatrix} 2915 & 3833 & 1199 \\ 330 & 285 & 450 \\ 129 & 121 & 366 \end{bmatrix}$	0	0.86	0.37	0.30
						1	0.07	0.27	
						2	0.18	0.59	
	Optuna	NearMiss2	x	*criterion* = *entropy* *max depth* = 50 *max features* = 6 *min samples leaf* = 2 *min samples split* = 2 *n estimators* = 10	$\begin{bmatrix} 1222 & 4921 & 1804 \\ 561 & 401 & 103 \\ 322 & 121 & 173 \end{bmatrix}$	0	0.58	0.15	0.16
						1	0.07	0.38	
						2	0.08	0.28	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 70 *max features* = 7 *min samples leaf* = 1 *min samples split* = 8 *n estimators* = 78	$\begin{bmatrix} 7846 & 18 & 83 \\ 990 & 38 & 37 \\ 493 & 13 & 110 \end{bmatrix}$	0	0.84	0.99	0.41
						1	0.55	0.04	
						2	0.48	0.18	
	Optuna	Random UnserSampler	x	*criterion* = *gini* *max depth* = 100 *max features* = 1 *min samples leaf* = 4 *min samples split* = 4 *n estimators* = 42	$\begin{bmatrix} 7254 & 259 & 434 \\ 706 & 88 & 271 \\ 278 & 36 & 302 \end{bmatrix}$	0	0.88	0.91	0.46
						1	0.23	0.08	
						2	0.30	0.49	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 10 *max features* = 8 *min samples leaf* = 3 *min samples split* = 8 *n estimators* = 67	$\begin{bmatrix} 6926 & 809 & 212 \\ 868 & 158 & 39 \\ 427 & 46 & 143 \end{bmatrix}$	0	0.88	0.93	0.43
						1	0.27	0.07	
						2	0.36	0.23	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 60 *max features* = *auto* *min samples leaf* = 2 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 13166 & 43 & 56 \\ 1503 & 182 & 123 \\ 708 & 16 & 111 \end{bmatrix}$	0	0.86	0.99	0.43
						1	0.76	0.10	
						2	0.38	0.13	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 6 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 2 *n estimators* = 100	$\begin{bmatrix} 11387 & 1412 & 466 \\ 511 & 803 & 494 \\ 307 & 94 & 434 \end{bmatrix}$	0	0.93	0.84	0.56
						1	0.35	0.44	
						2	0.31	0.52	
	RandomizedSearch	NearMiss 1	x	*criterion* = *gini* *max depth* = 40 *max features* = *sqrt* *min samples leaf* = 2 *min samples split* = 6 *n estimators* = 110	$\begin{bmatrix} 5971 & 5470 & 1824 \\ 289 & 518 & 1001 \\ 221 & 39 & 575 \end{bmatrix}$	0	0.92	0.45	0.34
						1	0.09	0.29	
						2	0.17	0.69	
	RandomizedSearch	NearMiss 2	x	*criterion* = *gini* *max depth* = 30 *max features* = 2 *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 2723 & 6586 & 3956 \\ 667 & 635 & 506 \\ 431 & 227 & 177 \end{bmatrix}$	0	0.71	0.21	0.17
						1	0.09	0.35	
						2	0.04	0.21	
	RandomizedSearch	NearMiss 3	x	*criterion* = *gini* *max depth* = 60 *max features* = 7 *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 1000	$\begin{bmatrix} 12393 & 334 & 538 \\ 1075 & 128 & 605 \\ 550 & 5 & 280 \end{bmatrix}$	0	0.88	0.93	0.42
						1	0.27	0.07	
						2	0.20	0.34	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 60 *max features* = *sqrt* *min samples leaf* = 2 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 13076 & 113 & 76 \\ 1446 & 198 & 164 \\ 702 & 6 & 127 \end{bmatrix}$	0	0.86	0.99	0.44
						1	0.62	0.11	
						2	0.35	0.15	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 80 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 10 *n estimators* = 110	$\begin{bmatrix} 12403 & 440 & 422 \\ 747 & 396 & 665 \\ 421 & 8 & 406 \end{bmatrix}$	0	0.91	0.94	0.52
						1	0.47	0.22	
						2	0.27	0.49	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 20 *max features* = *sqrt* *min samples leaf* = 2 *min samples split* = 50 *n estimators* = 60	$\begin{bmatrix} 12497 & 581 & 187 \\ 1156 & 332 & 320 \\ 519 & 6 & 310 \end{bmatrix}$	0	0.88	0.94	0.51
						1	0.36	0.18	
						2	0.38	0.37	

Table 5.9: Experiments with Random Forest and Maximum Angles

Random Forest	Minimum Angles								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Optuna	x	x	*criterion* = *entropy* *max depth* = 40 *max features* = 4 *min samples leaf* = 3 *min samples split* = 6 *n estimators* = 12	$\begin{bmatrix} 7787 & 148 & 12 \\ 1039 & 11 & 15 \\ 547 & 8 & 61 \end{bmatrix}$	0	0.83	0.98	0.36
						1	0.07	0.01	
						2	0.69	0.10	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 1 *min samples leaf* = 4 *min samples split* = 6 *n estimators* = 125	$\begin{bmatrix} 7581 & 123 & 243 \\ 660 & 73 & 332 \\ 384 & 2 & 230 \end{bmatrix}$	0	0.88	0.95	0.45
						1	0.37	0.07	
						2	0.29	0.37	
	Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 20 *max features* = 4 *min samples leaf* = 3 *min samples split* = 2 *n estimators* = 130	$\begin{bmatrix} 2175 & 4066 & 1706 \\ 130 & 391 & 544 \\ 32 & 32 & 552 \end{bmatrix}$	0	0.93	0.27	0.30
						1	0.09	0.37	
						2	0.20	0.90	
	Optuna	NearMiss 2	x	*criterion* = *entropy* *max depth* = 10 *max features* = 1 *min samples leaf* = 5 *min samples split* = 6 *n estimators* = 75	$\begin{bmatrix} 1793 & 5112 & 1042 \\ 129 & 428 & 508 \\ 106 & 65 & 445 \end{bmatrix}$	0	0.88	0.23	0.28
						1	0.08	0.40	
						2	0.22	0.72	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 100 *max features* = 4 *min samples leaf* = 1 *min samples split* = 2 *n estimators* = 10	$\begin{bmatrix} 7884 & 35 & 28 \\ 1053 & 10 & 2 \\ 593 & 0 & 23 \end{bmatrix}$	0	0.83	0.99	0.33
						1	0.22	0.01	
						2	0.43	0.04	
	Optuna	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 70 *max features* = 1 *min samples leaf* = 3 *min samples split* = 8 *n estimators* = 160	$\begin{bmatrix} 7346 & 149 & 452 \\ 536 & 56 & 473 \\ 279 & 4 & 333 \end{bmatrix}$	0	0.90	0.92	0.45
						1	0.27	0.05	
						2	0.26	0.54	
	Optuna	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 100 *max features* = 2 *min samples leaf* = 4 *min samples split* = 10 *n estimators* = 19	$\begin{bmatrix} 7480 & 203 & 264 \\ 858 & 43 & 164 \\ 420 & 7 & 189 \end{bmatrix}$	0	0.85	0.94	0.42
						1	0.17	0.04	
						2	0.31	0.31	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 90 *max features* = 7 *min samples leaf* = 3 *min samples split* = 2 *n estimators* = 10	$\begin{bmatrix} 12962 & 252 & 51 \\ 1658 & 116 & 34 \\ 666 & 29 & 140 \end{bmatrix}$	0	0.85	0.98	0.43
						1	0.29	0.06	
						2	0.62	0.17	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 5 *min samples leaf* = 1 *min samples split* = 50 *n estimators* = 160	$\begin{bmatrix} 12322 & 766 & 177 \\ 1072 & 586 & 150 \\ 453 & 199 & 183 \end{bmatrix}$	0	0.89	0.93	0.51
						1	0.38	0.32	
						2	0.36	0.22	
	RandomizedSearch	NearMiss 1	x	*criterion* = *gini* *max depth* = *None* *max features* = 5 *min samples leaf* = 2 *min samples split* = 10 *n estimators* = 60	$\begin{bmatrix} 6448 & 4194 & 2623 \\ 453 & 719 & 636 \\ 123 & 173 & 539 \end{bmatrix}$	0	0.92	0.49	0.36
						1	0.14	0.40	
						2	0.14	0.65	
	RandomizedSearch	NearMiss 2	x	*criterion* = *entropy* *max depth* = 40 *max features* = 7 *min samples leaf* = 4 *min samples split* = 6 *n estimators* = 110	$\begin{bmatrix} 1619 & 7458 & 4188 \\ 182 & 792 & 834 \\ 72 & 237 & 526 \end{bmatrix}$	0	0.86	0.12	0.18
						1	0.09	0.44	
						2	0.09	0.63	
	RandomizedSearch	NearMiss 3	x	*criterion* = *gini* *max depth* = 50 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 10 *n estimators* = 310	$\begin{bmatrix} 10922 & 1738 & 605 \\ 1243 & 321 & 244 \\ 588 & 79 & 168 \end{bmatrix}$	0	0.86	0.82	0.39
						1	0.15	0.18	
						2	0.17	0.20	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 40 *max features* = 7 *min samples leaf* = 3 *min samples split* = 6 *n estimators* = 210	$\begin{bmatrix} 13208 & 56 & 1 \\ 1770 & 16 & 22 \\ 764 & 12 & 59 \end{bmatrix}$	0	0.84	1.00	0.35
						1	0.19	0.01	
						2	0.72	0.07	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *gini* *max depth* = 50 *max features* = *sqrt* *min samples leaf* = 2 *min samples split* = 2 *n estimators* = 460	$\begin{bmatrix} 12194 & 685 & 386 \\ 999 & 537 & 272 \\ 417 & 198 & 220 \end{bmatrix}$	0	0.90	0.92	0.50
						1	0.38	0.30	
						2	0.25	0.26	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 10 *max features* = 2 *min samples leaf* = 2 *min samples split* = 50 *n estimators* = 410	$\begin{bmatrix} 11504 & 1550 & 211 \\ 878 & 728 & 202 \\ 383 & 162 & 290 \end{bmatrix}$	0	0.90	0.87	0.53
						1	0.30	0.40	
						2	0.41	0.35	

Table 5.10: Experiments with Random Forest and Minimum Angles

	AREAS								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
Random Forest	Optuna	x	x	*criterion* = *entropy* *max depth* = 70 *max features* = 8 *min samples leaf* = 5 *min samples split* = 6 *n estimators* = 20	$\begin{bmatrix} 7922 & 21 & 4 \\ 929 & 136 & 0 \\ 572 & 19 & 25 \end{bmatrix}$	0	0.84	1.00	0.40
						1	0.77	0.13	
						2	0.86	0.04	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 1 *min samples leaf* = 2 *min samples split* = 6 *n estimators* = 77	$\begin{bmatrix} 7628 & 189 & 130 \\ 819 & 33 & 213 \\ 486 & 5 & 125 \end{bmatrix}$	0	0.85	0.96	0.40
						1	0.15	0.03	
						2	0.27	0.20	
	Optuna	NearMiss 1	x	*criterion* = *gini* *max depth* = 20 *max features* = 2 *min samples leaf* = 5 *min samples split* = 6 *n estimators* = 152	$\begin{bmatrix} 4472 & 2695 & 780 \\ 290 & 287 & 488 \\ 94 & 81 & 441 \end{bmatrix}$	0	0.93	0.27	0.41
						1	0.92	0.56	
						2	0.26	0.72	
	Optuna	NearMiss 2	x	*criterion* = *entropy* *max depth* = 60 *max features* = 1 *min samples leaf* = 3 *min samples split* = 12 *n estimators* = 130	$\begin{bmatrix} 3579 & 1391 & 2977 \\ 109 & 68 & 888 \\ 69 & 50 & 497 \end{bmatrix}$	0	0.95	0.45	0.29
						1	0.05	0.06	
						2	0.11	0.81	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 30 *max features* = 7 *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 16	$\begin{bmatrix} 7899 & 35 & 13 \\ 992 & 71 & 2 \\ 551 & 0 & 65 \end{bmatrix}$	0	0.84	0.99	0.41
						1	0.67	0.07	
						2	0.81	0.11	
	Optuna	Random UnserSampler	x	*criterion* = *gini* *max depth* = 50 *max features* = 6 *min samples leaf* = 1 *min samples split* = 8 *n estimators* = 74	$\begin{bmatrix} 7483 & 342 & 122 \\ 754 & 215 & 96 \\ 449 & 92 & 75 \end{bmatrix}$	0	0.86	0.94	0.44
						1	0.33	0.20	
						2	0.26	0.12	
	Optuna	Cluster Centroids	x	*criterion* = *gini* *max depth* = 40 *max features* = 7 *min samples leaf* = 3 *min samples split* = 6 *n estimators* = 37	$\begin{bmatrix} 7578 & 330 & 39 \\ 801 & 151 & 113 \\ 553 & 25 & 38 \end{bmatrix}$	0	0.85	0.95	0.39
						1	0.30	0.14	
						2	0.20	0.06	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 60 *max features* = *auto* *min samples leaf* = 2 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 12331 & 396 & 538 \\ 1633 & 79 & 96 \\ 481 & 45 & 309 \end{bmatrix}$	0	0.85	0.93	0.44
						1	0.15	0.04	
						2	0.33	0.37	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = *None* *max features* = 4 *min samples leaf* = 3 *min samples split* = 50 *n estimators* = 160	$\begin{bmatrix} 12653 & 98 & 514 \\ 1344 & 130 & 334 \\ 347 & 9 & 479 \end{bmatrix}$	0	0.88	0.95	0.50
						1	0.55	0.07	
						2	0.36	0.57	
	RandomizedSearch	NearMiss 1	x	*criterion* = *entropy* *max depth* = 20 *max features* = *sqrt* *min samples leaf* = 2 *min samples split* = 50 *n estimators* = 60	$\begin{bmatrix} 6162 & 2849 & 4254 \\ 288 & 402 & 1118 \\ 45 & 119 & 671 \end{bmatrix}$	0	0.95	0.46	0.32
						1	0.12	0.22	
						2	0.11	0.80	
	RandomizedSearch	NearMiss 2	x	*criterion* = *gini* *max depth* = 40 *max features* = 1 *min samples leaf* = 4 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 4030 & 5470 & 3765 \\ 151 & 357 & 1300 \\ 60 & 114 & 661 \end{bmatrix}$	0	0.95	0.30	0.25
						1	0.06	0.20	
						2	0.12	0.79	
	RandomizedSearch	NearMiss 3	x	*criterion* = *gini* *max depth* = 90 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 60	$\begin{bmatrix} 7609 & 1543 & 4113 \\ 433 & 317 & 1058 \\ 70 & 8 & 757 \end{bmatrix}$	0	0.94	0.57	0.37
						1	0.17	0.18	
						2	0.13	0.91	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 40 *max features* = *sqrt* *min samples leaf* = 3 *min samples split* = 20 *n estimators* = 60	$\begin{bmatrix} 12505 & 515 & 245 \\ 1690 & 2 & 116 \\ 482 & 1 & 352 \end{bmatrix}$	0	0.85	0.94	0.45
						1	0.00	0.00	
						2	0.49	0.42	
	RandomizedSearch	Random UnderSampler	x	*criterion* = *entropy* *max depth* = 80 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 10 *n estimators* = 110	$\begin{bmatrix} 10920 & 519 & 1826 \\ 1011 & 212 & 585 \\ 193 & 9 & 633 \end{bmatrix}$	0	0.90	0.82	0.45
						1	0.29	0.12	
						2	0.21	0.76	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 90 *max features* = *sqrt* *min samples leaf* = 3 *min samples split* = 10 *n estimators* = 60	$\begin{bmatrix} 12245 & 464 & 556 \\ 1233 & 254 & 321 \\ 263 & 13 & 559 \end{bmatrix}$	0	0.89	0.92	0.53
						1	0.35	0.14	
						2	0.39	0.67	

Table 5.11: Experiments with Random Forest and Areas

Random Forest	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Distances								
	Patient-Wise Splitting 1								
	Optuna	x	x	*criterion* = *gini* *max depth* = 60 *max features* = 6 *min samples leaf* = 3 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 7835 & 49 & 63 \\ 1023 & 24 & 18 \\ 541 & 10 & 65 \end{bmatrix}$	0	0 : 0.84	0 : 0.98	0.37
						1	0.15	0.02	
						2	0.45	0.11	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 8 *min samples leaf* = 5 *min samples split* = 4 *n estimators* = 87	$\begin{bmatrix} 7603 & 212 & 132 \\ 843 & 178 & 44 \\ 459 & 29 & 128 \end{bmatrix}$	0	0.85	0.96	0.47
						1	0.42	0.17	
						2	0.42	0.21	
	Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 10 *max features* = 6 *min samples leaf* = 4 *min samples split* = 6 *n estimators* = 169	$\begin{bmatrix} 2032 & 5018 & 897 \\ 51 & 764 & 250 \\ 165 & 169 & 282 \end{bmatrix}$	0	0.90	0.26	0.30
						1	0.13	0.72	
						2	0.20	0.46	
	Optuna	NearMiss 2	x	*criterion* = *gini* *max depth* = 20 *max features* = 1 *min samples leaf* = 4 *min samples split* = 4 *n estimators* = 17	$\begin{bmatrix} 377 & 1305 & 6265 \\ 36 & 444 & 585 \\ 18 & 96 & 502 \end{bmatrix}$	0	0.87	0.05	0.17
						1	0.24	0.42	
						2	0.07	0.81	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 70 *max features* = 6 *min samples leaf* = 2 *min samples split* = 12 *n estimators* = 15	$\begin{bmatrix} 7854 & 32 & 61 \\ 1036 & 21 & 8 \\ 590 & 1 & 25 \end{bmatrix}$	0	0.83	0.99	0.34
						1	0.39	0.02	
						2	0.27	0.04	
	Optuna	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 90 *max features* = 7 *min samples leaf* = 4 *min samples split* = 4 *n estimators* = 82	$\begin{bmatrix} 7481 & 246 & 220 \\ 862 & 146 & 57 \\ 417 & 33 & 166 \end{bmatrix}$	0	0.85	0.94	0.47
						1	0.34	0.14	
						2	0.37	0.27	
	Optuna	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 80 *max features* = 8 *min samples leaf* = 5 *min samples split* = 6 *n estimators* = 64	$\begin{bmatrix} 7595 & 175 & 177 \\ 961 & 59 & 45 \\ 471 & 30 & 115 \end{bmatrix}$	0	0.84	0.96	0.41
						1	0.22	0.06	
						2	0.34	0.19	
	Patient-Wise Spliting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 60 *max features* = *auto* *min samples leaf* = 2 *min samples split* = 4 *n estimators* = 10	$\begin{bmatrix} 12331 & 396 & 538 \\ 1633 & 79 & 96 \\ 481 & 45 & 309 \end{bmatrix}$	0	0.85	0.93	0.44
						1	0.15	0.04	
						2	0.33	0.37	
	RandomizedSearch	x	✓	*criterion* = *gini* *max depth* = 10 *max features* = 4 *min samples leaf* = 2 *min samples split* = 50 *n estimators* = 160	$\begin{bmatrix} 12660 & 370 & 235 \\ 885 & 390 & 533 \\ 514 & 18 & 303 \end{bmatrix}$	0	0.90	0.95	0.52
						1	0.50	0.22	
						2	0.28	0.36	
	RandomizedSearch	NearMiss 1	x	*criterion* = *entropy* *max depth* = 10 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 110	$\begin{bmatrix} 7801 & 2721 & 2743 \\ 346 & 414 & 1048 \\ 65 & 145 & 625 \end{bmatrix}$	0	0.95	0.59	0.38
						1	0.13	0.23	
						2	0.14	0.75	
	RandomizedSearch	NearMiss 2	x	*criterion* = *gini* *max depth* = 10 *max features* = 3 *min samples leaf* = 2 *min samples split* = 4 *n estimators* = 60	$\begin{bmatrix} 3278 & 5495 & 4492 \\ 117 & 695 & 996 \\ 82 & 39 & 714 \end{bmatrix}$	0	0.95	0.59	0.26
						1	0.13	0.23	
						2	0.12	0.86	
	RandomizedSearch	NearMiss 3	x	*criterion* = *gini* *max depth* = 80 *max features* = 7 *min samples leaf* = 2 *min samples split* = 8 *n estimators* = 55	$\begin{bmatrix} 9992 & 740 & 2533 \\ 783 & 202 & 823 \\ 172 & 26 & 637 \end{bmatrix}$	0	0.91	0.75	0.41
						1	0.21	0.11	
						2	0.16	0.76	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 80 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 10 *n estimators* = 110	$\begin{bmatrix} 12756 & 0 & 509 \\ 1661 & 3 & 144 \\ 545 & 1 & 289 \end{bmatrix}$	0	0.85	0.96	0.41
						1	0.75	0.00	
						2	0.31	0.35	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 10 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 110	$\begin{bmatrix} 10762 & 1070 & 1433 \\ 981 & 263 & 564 \\ 182 & 78 & 575 \end{bmatrix}$	0	0.90	0.81	0.45
						1	0.19	0.15	
						2	0.22	0.69	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 10 *max features* = 6 *min samples leaf* = 4 *min samples split* = 8 *n estimators* = 130	$\begin{bmatrix} 10774 & 1113 & 1378 \\ 1047 & 311 & 450 \\ 278 & 35 & 522 \end{bmatrix}$	0	0.89	0.81	0.46
						1	0.21	0.17	
						2	0.72	0.63	

Table 5.12: Experiments with Random Forest and Distances

Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
Hog Features								
Patient-Wise Splitting 1								
Optuna	x	x	*criterion* = *entropy* *max depth* = 70 *max features* = 5 *min samples leaf* = 3 *min samples split* = 2 *n estimators* = 10	$\begin{bmatrix} 7928 & 17 & 2 \\ 1065 & 0 & 149 \\ 613 & 2 & 1 \end{bmatrix}$	0 1 2	0.83 0.00 0.33	1.00 0.00 0.00	0.30
Optuna	x	✓	*criterion* = *gini* *max depth* = 10 *max features* = 7 *min samples leaf* = 4 *min samples split* = 6 *n estimators* = 21	$\begin{bmatrix} 7859 & 79 & 9 \\ 1056 & 1 & 8 \\ 603 & 4 & 9 \end{bmatrix}$	0 1 2	0.83 0.01 0.35	0.99 0.00 0.01	0.31
Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 10 *max features* = 7 *min samples leaf* = 3 *min samples split* = 12 *n estimators* = 184	$\begin{bmatrix} 1573 & 2179 & 4195 \\ 142 & 212 & 711 \\ 7 & 53 & 556 \end{bmatrix}$	0 1 2	0.91 0.09 0.10	0.20 0.20 0.90	0.21
Optuna	NearMiss 2	x	*criterion* = *gini* *max depth* = 100 *max features* = 1 *min samples leaf* = 2 *min samples split* = 10 *n estimators* = 10	$\begin{bmatrix} 1468 & 2234 & 4245 \\ 184 & 243 & 638 \\ 202 & 100 & 314 \end{bmatrix}$	0 1 2	0.79 0.09 0.06	0.18 0.23 0.51	0.18
Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 100 *max features* = 10 *min samples leaf* = 3 *min samples split* = 6 *n estimators* = 10	$\begin{bmatrix} 7925 & 13 & 9 \\ 1064 & 1 & 0 \\ 611 & 3 & 2 \end{bmatrix}$	0 1 2	0.83 0.06 0.18	1.00 0.00 0.00	0.30
Optuna	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 100 *max features* = 7 *min samples leaf* = 4 *min samples split* = 2 *n estimators* = 96	$\begin{bmatrix} 7801 & 107 & 39 \\ 1010 & 30 & 25 \\ 530 & 17 & 69 \end{bmatrix}$	0 1 2	0.84 0.19 0.52	0.98 0.03 0.11	0.38
Patient-Wise Splitting 2								
RandomisedSearch	x	x	*criterion* = *enyropy* *max depth* = 30 *max features* = 6 *min samples leaf* = 1 *min samples split* = 8 *n estimators* = 30	$\begin{bmatrix} 13265 & 0 & 0 \\ 1808 & 0 & 0 \\ 835 & 0 & 0 \end{bmatrix}$	0 1 2	0.83 0.00 0.00	1.00 0.00 0.00	0.30
RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 10 *max features* = 7 *min samples leaf* = 3 *min samples split* = 4 *n estimators* = 26	$\begin{bmatrix} 13136 & 81 & 48 \\ 1757 & 41 & 10 \\ 770 & 19 & 46 \end{bmatrix}$	0 1 2	0.84 0.29 0.44	0.99 0.02 0.06	0.35
RandomizedSearch	NearMiss 1	x	*criterion* = *entropy* *max depth* = 50 *max features* = 5 *min samples leaf* = 2 *min samples split* = 2 *n estimators* = 13	$\begin{bmatrix} 4314 & 4355 & 4596 \\ 441 & 519 & 848 \\ 144 & 151 & 540 \end{bmatrix}$	0 1 2	0.88 0.10 0.09	0.33 0.29 0.65	0.26
RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 30 *max features* = 8 *min samples leaf* = 4 *min samples split* = 6 *n estimators* = 10	$\begin{bmatrix} 13250 & 3 & 12 \\ 1767 & 14 & 27 \\ 833 & 0 & 2 \end{bmatrix}$	0 1 2	0.84 0.82 0.05	1.00 0.01 0.00	0.31
RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 10 *max features* = *sqrt* *min samples leaf* = 1 *min samples split* = 4 *n estimators* = 110	$\begin{bmatrix} 12821 & 312 & 132 \\ 1346 & 138 & 324 \\ 558 & 15 & 262 \end{bmatrix}$	0 1 2	0.87 0.30 0.36	0.97 0.08 0.31	0.46

Table 5.13: Experiments with Random Forest and Hog Features

Decision Trees	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Maximum Angles								
	Patient-Wise Splitting 1								
	Optuna	x	x	*criterion* = *gini* *max depth* = 6 *max features* = 5 *min samples leaf* = 3 *min samples split* = 2 *max leaf nodes* = 20	$\begin{bmatrix} 7285 & 546 & 116 \\ 1025 & 7 & 33 \\ 454 & 124 & 38 \end{bmatrix}$	0	0.83	0.92	0.32
						1	1 : 0.01	1 : 0.01	
						2	0.20	0.06	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 4 *max features* = *None* *min samples leaf* = 1 *min samples split* = 3 *max leaf nodes* = *None*	$\begin{bmatrix} 3773 & 2116 & 2058 \\ 217 & 347 & 501 \\ 115 & 85 & 416 \end{bmatrix}$	0	0.92	0.47	0.35
						1	0.14	0.33	
						2	0.14	0.68	
	Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 2 *max features* = 2 *min samples leaf* = 3 *min samples split* = 2 *max leaf nodes* = 20	$\begin{bmatrix} 1366 & 5341 & 1240 \\ 65 & 451 & 549 \\ 45 & 142 & 429 \end{bmatrix}$	0	0.93	0.17	0.24
						1	0.08	0.42	
						2	0.19	0.70	
	Optuna	NearMiss 2	x	*criterion* = *gini* *max depth* = 4 *max features* = 2 *min samples leaf* = 5 *min samples split* = 2 *max leaf nodes* = 10	$\begin{bmatrix} 837 & 5675 & 1435 \\ 309 & 491 & 265 \\ 194 & 155 & 267 \end{bmatrix}$	0	0.62	0.11	0.17
						1	0.08	0.46	
						2	0.14	0.43	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 5 *max features* = 9 *min samples leaf* = 1 *min samples split* = 2 *max leaf nodes* = 50	$\begin{bmatrix} 7371 & 511 & 65 \\ 892 & 167 & 6 \\ 465 & 77 & 74 \end{bmatrix}$	0	0.84	0.93	0.42
						1	0.22	0.16	
						2	0.51	0.12	
	Optuna	Random UnderSampler	x	*criterion* = *entropy* *max depth* = 7 *max features* = 9 *min samples leaf* = 9 *min samples split* = 2 *max leaf nodes* = 20	$\begin{bmatrix} 4097 & 2568 & 1282 \\ 153 & 655 & 257 \\ 176 & 196 & 244 \end{bmatrix}$	0	0.93	0.52	0.39
						1	0.19	0.62	
						2	0.14	0.40	
	Optuna	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 5 *max features* = 3 *min samples leaf* = 2 *min samples split* = 2 *max leaf nodes* = 40	$\begin{bmatrix} 5215 & 1459 & 1273 \\ 866 & 29 & 170 \\ 280 & 17 & 319 \end{bmatrix}$	0	0.82	0.66	0.34
						1	0.02	0.03	
						2	0.18	0.52	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 8 *max features* = 4 *min samples leaf* = 1 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 11996 & 704 & 565 \\ 1198 & 416 & 194 \\ 678 & 74 & 83 \end{bmatrix}$	0	0.86	0.90	0.42
						1	0.35	0.23	
						2	0.10	0.10	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 7 *max features* = 4 *min samples leaf* = 4 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 9088 & 2332 & 1845 \\ 373 & 665 & 770 \\ 318 & 32 & 485 \end{bmatrix}$	0	0.93	0.69	0.44
						1	0.22	0.37	
						2	0.16	0.58	
	RandomizedSearch	NearMiss 1	x	*criterion* = *gini* *max depth* = 5 *max features* = 7 *min samples leaf* = 5 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 8445 & 3098 & 1722 \\ 402 & 609 & 797 \\ 328 & 80 & 427 \end{bmatrix}$	0	0.92	0.64	0.40
						1	0.16	0.34	
						2	0.14	0.51	
	RandomizedSearch	NearMiss 2	x	*criterion* = *entropy* *max depth* = 4 *max features* = 1 *min samples leaf* = 5 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 2203 & 5450 & 5612 \\ 493 & 635 & 680 \\ 307 & 389 & 139 \end{bmatrix}$	0	0.73	0.17	0.15
						1	0.10	0.35	
						2	0.02	0.17	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 5 *max features* = 6 *min samples leaf* = 6 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 11967 & 1149 & 149 \\ 1029 & 508 & 271 \\ 646 & 102 & 87 \end{bmatrix}$	0	0.88	0.90	0.43
						1	0.29	0.28	
						2	0.17	0.10	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 5 *max features* = 6 *min samples leaf* = 6 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 8859 & 3740 & 666 \\ 341 & 1136 & 331 \\ 312 & 174 & 349 \end{bmatrix}$	0	0.93	0.67	0.48
						1	0.22	0.63	
						2	0.26	0.42	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *gini* *max depth* = 7 *max features* = 6 *min samples leaf* = 4 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 8266 & 4175 & 824 \\ 655 & 656 & 497 \\ 394 & 165 & 276 \end{bmatrix}$	0	0.89	0.62	0.38
						1	0.13	0.36	
						2	0.17	0.33	

Table 5.14: Experiments with Decision Trees and Maximum Angles

Decision Trees	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
Minimum Angles									
Patient-Wise Splitting 1									
	Optuna	x	x	*criterion* = *gini* *max depth* = 6 *max features* = 5 *min samples leaf* = 5 *max leaf nodes* = 20	$\begin{bmatrix} 7846 & 17 & 84 \\ 1055 & 10 & 0 \\ 494 & 41 & 81 \end{bmatrix}$	0 1 2	0.84 0.15 0.49	0.99 0.01 0.13	0.38
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 8 *max features* = 2 *min samples leaf* = 2 *max leaf nodes* = 60	$\begin{bmatrix} 5324 & 1405 & 1218 \\ 361 & 372 & 332 \\ 189 & 48 & 379 \end{bmatrix}$	0 1 2	0.91 0.20 0.10	0.67 0.35 0.52	0.32
	Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 8 *max features* = 9 *min samples leaf* = 5 *max leaf nodes* = 100	$\begin{bmatrix} 3179 & 2281 & 2487 \\ 331 & 463 & 271 \\ 213 & 82 & 321 \end{bmatrix}$	0 1 2	0.85 0.16 0.10	0.40 0.43 0.52	0.32
	Optuna	NearMiss 2	x	*criterion* = *entropy* *max depth* = 4 *max features* = 5 *min samples leaf* = 4 *max leaf nodes* = 20	$\begin{bmatrix} 1510 & 4860 & 1577 \\ 96 & 482 & 487 \\ 40 & 200 & 376 \end{bmatrix}$	0 1 2	0.92 0.09 0.15	0.19 0.45 0.61	0.24
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 8 *max features* = 7 *min samples leaf* = 8 *max leaf nodes* = 90	$\begin{bmatrix} 7269 & 323 & 355 \\ 872 & 105 & 88 \\ 449 & 42 & 125 \end{bmatrix}$	0 1 2	0.85 0.22 0.22	0.91 0.10 0.20	0.41
	Optuna	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 4 *max features* = 5 *min samples leaf* = 5 *max leaf nodes* = 80	$\begin{bmatrix} 4287 & 2457 & 1203 \\ 242 & 320 & 503 \\ 121 & 137 & 358 \end{bmatrix}$	0 1 2	0.92 0.11 0.17	0.54 0.30 0.58	0.37
	Optuna	Random UnderSampler	x	*criterion* = *entropy* *max depth* = 6 *max features* = 1 *min samples leaf* = 2 *max leaf nodes* = 70	$\begin{bmatrix} 4772 & 1790 & 1385 \\ 398 & 455 & 212 \\ 48 & 221 & 347 \end{bmatrix}$	0 1 2	0.91 0.18 0.18	0.60 0.43 0.56	0.42
Patient-Wise Splitting 2									
	RandomisedSearch	x	x	*criterion* = *entropy* *max depth* = 6 *max features* = 3 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 12698 & 385 & 182 \\ 1364 & 252 & 192 \\ 553 & 65 & 217 \end{bmatrix}$	0 1 2	0.87 0.36 0.37	0.96 0.14 0.26	0.47
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 3 *max features* = 2 *min samples leaf* = 4 *max leaf nodes* = *None*	$\begin{bmatrix} 5866 & 5682 & 1717 \\ 238 & 1011 & 559 \\ 42 & 392 & 401 \end{bmatrix}$	0 1 2	0.95 0.14 0.15	0.44 0.56 0.48	0.35
	RandomizedSearch	NearMiss 1	x	*criterion* = *gini* *max depth* = 2 *max features* = 4 *min samples leaf* = 7 *max leaf nodes* = *None*	$\begin{bmatrix} 4737 & 7510 & 1018 \\ 526 & 952 & 330 \\ 155 & 345 & 335 \end{bmatrix}$	0 1 2	0.87 0.11 0.20	0.36 0.53 0.40	0.32
	RandomizedSearch	NearMiss 2	x	*criterion* = *gini* *max depth* = 2 *max features* = 7 *min samples leaf* = 1 *min samples split* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 2000 & 5725 & 5540 \\ 250 & 679 & 879 \\ 111 & 158 & 566 \end{bmatrix}$	0 1 2	0.85 0.10 0.08	0.15 0.38 0.68	0.19
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 7 *max features* = 6 *min samples leaf* = 7 *max leaf nodes* = *None*	$\begin{bmatrix} 12361 & 638 & 266 \\ 1285 & 466 & 57 \\ 718 & 30 & 87 \end{bmatrix}$	0 1 2	0.86 0.41 0.21	0.93 0.26 0.10	0.45
	RandomizedSearch	Random UnserSampler	x	*criterion* = *gini* *max depth* = 4 *max features* = 3 *min samples leaf* = 1 *max leaf nodes* = *None*	$\begin{bmatrix} 8894 & 978 & 3393 \\ 515 & 483 & 810 \\ 114 & 177 & 544 \end{bmatrix}$	0 1 2	0.93 0.29 0.11	0.67 0.27 0.65	0.42
	RandomizedSearch	Cluster Centroids	x	*criterion* = *gini* *max depth* = 6 *max features* = 6 *min samples leaf* = 3 *max leaf nodes* = *None*	$\begin{bmatrix} 7544 & 4210 & 1511 \\ 989 & 522 & 297 \\ 275 & 195 & 365 \end{bmatrix}$	0 1 2	0.86 0.11 0.17	0.57 0.29 0.44	0.36

Table 5.15: Experiments with Decision Trees and Minimum Angles

Decision Trees	Areas								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Optuna	x	x	*criterion* = *gini* *max depth* = 7 *max features* = 5 *min samples leaf* = 1 *max leaf nodes* = 10	$\begin{bmatrix} 7899 & 23 & 25 \\ 1053 & 2 & 10 \\ 530 & 12 & 74 \end{bmatrix}$	0	0.83	0.99	0.37
						1	0.05	0.00	
						2	0.68	0.12	
	Optuna	x	✓	*criterion* = *gini* *max depth* = 3 *max features* = 4 *min samples leaf* = 9 *max leaf nodes* = 20	$\begin{bmatrix} 4655 & 1906 & 1386 \\ 134 & 173 & 758 \\ 61 & 89 & 466 \end{bmatrix}$	0	0.96	0.59	0.37
						1	0.08	0.16	
						2	0.18	0.76	
	Optuna	NearMiss 1	x	*criterion* = *gini* *max depth* = 5 *max features* = 5 *min samples leaf* = 4 *max leaf nodes* = 90	$\begin{bmatrix} 2577 & 4278 & 1092 \\ 121 & 469 & 475 \\ 145 & 92 & 379 \end{bmatrix}$	0	0.91	0.32	0.31
						1	0.10	0.44	
						2	0.19	0.62	
	Optuna	NearMiss 2	x	*criterion* = *gini* *max depth* = 5 *max features* = 7 *min samples leaf* = 8 *max leaf nodes* = 10	$\begin{bmatrix} 1854 & 5486 & 607 \\ 8 & 920 & 137 \\ 8 & 328 & 280 \end{bmatrix}$	0	0.99	0.23	0.32
						1	0.14	0.86	
						2	0.27	0.45	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 7 *max features* = 5 *min samples leaf* = 8 *max leaf nodes* = 90	$\begin{bmatrix} 7012 & 643 & 292 \\ 810 & 227 & 28 \\ 470 & 66 & 80 \end{bmatrix}$	0	0.85	0.88	0.42
						1	0.24	0.21	
						2	0.20	0.13	
	Optuna	Random UnderSampler	x	*criterion* = *gini* *max depth* = 5 *max features* = 1 *min samples leaf* = 6 *max leaf nodes* = 40	$\begin{bmatrix} 3744 & 3247 & 956 \\ 179 & 686 & 200 \\ 78 & 174 & 364 \end{bmatrix}$	0	0.94	0.47	0.41
						1	0.17	0.64	
						2	0.24	0.59	
	Optuna	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 2 *max features* = 7 *min samples leaf* = 6 *max leaf nodes* = 30	$\begin{bmatrix} 178 & 6158 & 1611 \\ 182 & 273 & 610 \\ 36 & 201 & 379 \end{bmatrix}$	0	0.45	0.02	0.12
						1	0.04	0.26	
						2	0.15	0.62	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 4 *max features* = 6 *min samples leaf* = 6 *max leaf nodes* = *None*	$\begin{bmatrix} 12436 & 628 & 201 \\ 1265 & 280 & 263 \\ 462 & 157 & 216 \end{bmatrix}$	0	0.88	0.94	0.46
						1	0.26	0.15	
						2	0.32	0.26	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 6 *max features* = 8 *min samples leaf* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 8249 & 3407 & 1609 \\ 249 & 1028 & 531 \\ 145 & 236 & 454 \end{bmatrix}$	0	0.95	0.62	0.45
						1	0.22	0.57	
						2	0.18	0.54	
	RandomizedSearch	NearMiss 1	x	*criterion* = *gini* *max depth* = 2 *max features* = 6 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 10154 & 1512 & 1599 \\ 1017 & 2 & 789 \\ 271 & 63 & 501 \end{bmatrix}$	0	0.89	0.77	0.36
						1	0.00	0.00	
						2	0.17	0.60	
	RandomizedSearch	NearMiss 2	x	*criterion* = *entropy* *max depth* = 4 *max features* = 5 *min samples leaf* = 7 *max leaf nodes* = *None*	$\begin{bmatrix} 5182 & 5770 & 2313 \\ 417 & 265 & 1126 \\ 65 & 97 & 673 \end{bmatrix}$	0	0.91	0.39	0.30
						1	0.04	0.15	
						2	0.16	0.81	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 6 *max features* = 4 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 11905 & 733 & 627 \\ 1477 & 276 & 55 \\ 517 & 145 & 173 \end{bmatrix}$	0	0.86	0.90	0.42
						1	0.24	0.15	
						2	0.20	0.21	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 7 *max features* = 7 *min samples leaf* = 7 *max leaf nodes* = *None*	$\begin{bmatrix} 6766 & 3340 & 3159 \\ 506 & 296 & 1006 \\ 157 & 184 & 494 \end{bmatrix}$	0	0.91	0.51	0.31
						1	0.08	0.16	
						2	0.11	0.59	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *gini* *max depth* = 6 *max features* = 6 *min samples leaf* = 2 *max leaf nodes* = *None*	$\begin{bmatrix} 8080 & 2454 & 2731 \\ 718 & 415 & 675 \\ 330 & 97 & 408 \end{bmatrix}$	0	0.89	0.61	0.36
						1	0.14	0.23	
						2	0.11	0.49	

Table 5.16: Experiments with Decisio Trees and Areas

Decision Trees	Distances								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Optuna	x	x	*criterion* = *gini* *max depth* = 6 *max features* = 4 *min samples leaf* = 6 *max leaf nodes* = 80	$\begin{bmatrix} 7713 & 82 & 152 \\ 1062 & 0 & 3 \\ 550 & 49 & 17 \end{bmatrix}$	0	0.83	0.97	0.40
						1	0.00	0.00	
						2	0.30	0.22	
	Optuna	x	✓	*criterion* = *entropy* *max depth* = 7 *max features* = 5 *min samples leaf* = 7 *min samples split* = 2 *max leaf nodes* = 90	$\begin{bmatrix} 4641 & 3007 & 299 \\ 334 & 675 & 56 \\ 289 & 175 & 152 \end{bmatrix}$	0	0.88	0.58	0.42
						1	0.18	0.63	
						2	0.30	0.25	
	Optuna	NearMiss 1	x	*criterion* = *entropy* *max depth* = 4 *max features* = 6 *min samples leaf* = 8 *max leaf nodes* = 40	$\begin{bmatrix} 1494 & 6170 & 283 \\ 309 & 709 & 47 \\ 16 & 470 & 130 \end{bmatrix}$	0	0.82	0.19	0.24
						1	0.10	0.67	
						2	0.28	0.21	
	Optuna	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 6 *max features* = 6 *min samples leaf* = 5 *max leaf nodes* = 30	$\begin{bmatrix} 7465 & 446 & 36 \\ 1003 & 44 & 18 \\ 545 & 22 & 49 \end{bmatrix}$	0	0.83	0.94	0.36
						1	0.09	0.04	
						2	0.48	0.08	
	Optuna	Random UnderSampler	x	*criterion* = *gini* *max depth* = 3 *max features* = 9 *min samples leaf* = 1 *min samples split* = 2 *max leaf nodes* = 30	$\begin{bmatrix} 5694 & 2077 & 176 \\ 542 & 473 & 50 \\ 432 & 70 & 114 \end{bmatrix}$	0	0.85	0.72	0.42
						1	0.18	0.44	
						2	0.34	0.19	
	Optuna	Cluster Centroids	x	*criterion* = *gini* *max depth* = 4 *max features* = 8 *min samples leaf* = 7 *max leaf nodes* = 90	$\begin{bmatrix} 6929 & 825 & 193 \\ 695 & 695 & 52 \\ 393 & 117 & 106 \end{bmatrix}$	0	0.86	0.87	0.45
						1	0.25	0.30	
						2	0.30	0.17	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *gini* *max depth* = 6 *max features* = 5 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 11629 & 1471 & 165 \\ 1638 & 36 & 134 \\ 455 & 65 & 315 \end{bmatrix}$	0	0.85	0.88	0.44
						1	0.02	0.02	
						2	0.51	0.38	
	RandomizedSearch	x	✓	*criterion* = *entropy* *max depth* = 5 *max features* = 7 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 7538 & 4355 & 1372 \\ 636 & 780 & 392 \\ 197 & 239 & 399 \end{bmatrix}$	0	0.90	0.57	0.39
						1	0.15	0.43	
						2	0.18	0.48	
	RandomizedSearch	NearMiss 1	x	*criterion* = *entropy* *max depth* = 4 *max features* = 6 *min samples leaf* = 6 *max leaf nodes* = *None*	$\begin{bmatrix} 5642 & 3111 & 4512 \\ 737 & 718 & 353 \\ 108 & 441 & 286 \end{bmatrix}$	0	0.87	0.43	0.30
						1	0.17	0.40	
						2	0.06	0.34	
	RandomizedSearch	NearMiss 2	x	*criterion* = *entropy* *max depth* = 1 *max features* = 2 *min samples leaf* = 8 *max leaf nodes* = *None*	$\begin{bmatrix} 5484 & 0 & 7781 \\ 303 & 0 & 1505 \\ 128 & 0 & 707 \end{bmatrix}$	0	0.93	0.41	0.23
						1	0.00	0.00	
						2	0.07	0.85	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *gini* *max depth* = 7 *max features* = 6 *min samples leaf* = 6 *max leaf nodes* = *None*	$\begin{bmatrix} 10853 & 1877 & 535 \\ 1274 & 289 & 245 \\ 451 & 243 & 141 \end{bmatrix}$	0	0.86	0.82	0.38
						1	0.12	0.16	
						2	0.15	0.17	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 5 *max features* = 5 *min samples leaf* = 1 *max leaf nodes* = *None*	$\begin{bmatrix} 6994 & 4957 & 1314 \\ 473 & 886 & 449 \\ 142 & 471 & 222 \end{bmatrix}$	0	0.92	0.53	0.35
						1	0.14	0.49	
						2	0.11	0.27	
	RandomizedSearch	Cluster Centroids	x	*criterion* = *entropy* *max depth* = 5 *max features* = 8 *min samples leaf* = 8 *max leaf nodes* = *None*	$\begin{bmatrix} 6229 & 6448 & 588 \\ 187 & 1265 & 356 \\ 73 & 599 & 163 \end{bmatrix}$	0	0.96	0.47	0.35
						1	0.15	0.70	
						2	0.15	0.20	

Table 5.17: Experiments with Decision Trees and Distances

Decision Trees	Hog Features								
	Patient-Wise Splitting 1								
	Search Method	Imbalance Method	Cost Sensitive	Parameters	Confusion Matrix	Class	Precision	Recall	F1 macro
	Optuna	x	x	*criterion* = *gini* *max depth* = 6 *max features* = 4 *min samples leaf* = 6 *max leaf nodes* = 80	$\begin{bmatrix} 7713 & 82 & 152 \\ 1062 & 0 & 3 \\ 550 & 49 & 17 \end{bmatrix}$	0	0.83	0.97	0.40
						1	0.00	0.00	
						2	0.30	0.22	
	Optuna	x	✓	*criterion* = *gini* *max depth* = 1 *max features* = 4 *min samples leaf* = 6 *min samples split* = 2 *max leaf nodes* = 90	$\begin{bmatrix} 0 & 6668 & 1279 \\ 0 & 979 & 86 \\ 0 & 483 & 133 \end{bmatrix}$	0	0.00	0.00	0.42
						1	0.12	0.92	
						2	0.09	0.22	
	Optuna	NearMiss 1	x	*criterion* = *gini* *max depth* = 2 *max features* = 8 *min samples leaf* = 2 *max leaf nodes* = 10	$\begin{bmatrix} 1201 & 4011 & 2735 \\ 237 & 633 & 195 \\ 150 & 294 & 172 \end{bmatrix}$	0	0.76	0.15	0.19
						1	0.13	0.59	
						2	0.06	0.28	
	Optuna	NearMiss 2	x	*criterion* = *gini* *max depth* = 1 *max features* = 2 *min samples leaf* = 5 *max leaf nodes* = 50	$\begin{bmatrix} 5045 & 2902 & 0 \\ 889 & 176 & 0 \\ 279 & 337 & 0 \end{bmatrix}$	0	0.81	0.63	0.26
						1	0.05	0.17	
						2	0.00	0.00	
	Optuna	Random UnderSampler	x	*criterion* = *entropy* *max depth* = 4 *max features* = 1 *min samples leaf* = 3 *min samples split* = 2 *max leaf nodes* = 10	$\begin{bmatrix} 2728 & 2609 & 2610 \\ 335 & 515 & 215 \\ 130 & 248 & 238 \end{bmatrix}$	0	0.78	0.27	0.28
						1	0.08	0.18	
						2	0.08	0.39	
	Optuna	Cluster Centroids	x	*criterion* = *gini* *max depth* = 4 *max features* = 8 *min samples leaf* = 7 *max leaf nodes* = 90	$\begin{bmatrix} 6929 & 825 & 193 \\ 695 & 695 & 52 \\ 393 & 117 & 106 \end{bmatrix}$	0	0.86	0.87	0.45
						1	0.25	0.30	
						2	0.30	0.17	
	Patient-Wise Splitting 2								
	RandomisedSearch	x	x	*criterion* = *entropy* *max depth* = 8 *max features* = 3 *min samples leaf* = 5 *max leaf nodes* = *None*	$\begin{bmatrix} 12860 & 275 & 130 \\ 1755 & 13 & 40 \\ 824 & 4 & 7 \end{bmatrix}$	0	0.83	0.97	0.31
						1	0.04	0.01	
						2	0.04	0.01	
	RandomizedSearch	x	✓	*criterion* = *gini* *max depth* = 5 *max features* = 7 *min samples leaf* = 1 *max leaf nodes* = *None*	$\begin{bmatrix} 5733 & 5661 & 1871 \\ 675 & 611 & 522 \\ 322 & 375 & 138 \end{bmatrix}$	0	0.85	0.43	0.27
						1	0.09	0.34	
						2	0.05	0.17	
	RandomizedSearch	NearMiss 1	x	*criterion* = *entropy* *max depth* = 2 *max features* = 3 *min samples leaf* = 3 *max leaf nodes* = *None*	$\begin{bmatrix} 7433 & 971 & 4861 \\ 4861 & 112 & 547 \\ 378 & 107 & 350 \end{bmatrix}$	0	0.83	0.56	0.28
						1	0.09	0.06	
						2	0.06	0.42	
	RandomizedSearch	NearMiss 2	x	*criterion* = *entropy* *max depth* = 4 *max features* = 1 *min samples leaf* = 7 *max leaf nodes* = *None*	$\begin{bmatrix} 6674 & 487 & 6104 \\ 827 & 37 & 944 \\ 510 & 10 & 315 \end{bmatrix}$	0	0.83	0.50	0.25
						1	0.07	0.02	
						2	0.04	0.38	
	RandomizedSearch	Neighbourhood Cleaning Rule	x	*criterion* = *entropy* *max depth* = 8 *max features* = 6 *min samples leaf* = 1 *max leaf nodes* = *None*	$\begin{bmatrix} 12405 & 516 & 344 \\ 1634 & 77 & 97 \\ 760 & 27 & 48 \end{bmatrix}$	0	0.84	0.94	0.34
						1	0.12	0.04	
						2	0.10	0.06	
	RandomizedSearch	Random UnserSampler	x	*criterion* = *entropy* *max depth* = 8 *max features* = 8 *min samples leaf* = 6 *max leaf nodes* = *None*	$\begin{bmatrix} 9511 & 1646 & 2108 \\ 981 & 490 & 337 \\ 471 & 109 & 255 \end{bmatrix}$	0	0.87	0.72	0.39
						1	0.22	0.27	
						2	0.09	0.31	

Table 5.18: Experiments with Decision Trees and Hog Features

LEAVE ONE SUBJECT OUT							
Model	Imbalance Method	Cost Sensitive	Parameters	Class	Precision	Recall	F1 macro
MAXIMUM ANGLES							
SVM	x	✓	$kernel = rbf$ $C = 1$ $gamma = 0.01$	0	0.95	0.75	0.44
				1	0.21	0.38	
				2	0.14	0.38	
Random Forest	x	✓	$criterion = entropy$ $max\ depth = 6$ $max\ features = sqrt$ $min\ samples\ leaf = 1$ $min\ samples\ split = 2$ $n\ estimators = 100$	0	0.91	0.85	0.51
				1	0.25	0.28	
				2	0.32	0.51	
Decision Tree	x	✓	$criterion = entropy$ $max\ depth = 7$ $max\ features = 4$ $min\ samples\ leaf = 4$ $min\ samples\ split = 2$ $max\ leaf\ nodes = None$	0	0.89	0.65	0.40
				1	0.15	0.36	
				2	0.18	0.36	
MINIMUM ANGLES							
SVM	Random UnserSampler	x	$kernel = rbf$ $C = 1$ $gamma = 0.01$	0	0.91	0.75	0.46
				1	0.21	0.29	
				2	0.22	0.54	
Random Forest	Cluster Centroids	x	$criterion = entropy$ $max\ depth = 10$ $max\ features = 2$ $min\ samples\ leaf = 2$ $min\ samples\ split = 50$ $n\ estimators = 410$	0	0.89	0.83	0.52
				1	0.20	0.32	
				2	0.31	0.43	
Decision Tree	Random UnderSampler	x	$criterion = gini$ $max\ depth = 4$ $max\ features = 3$ $min\ samples\ leaf = 1$ $max\ leaf\ nodes = None$	0	0.91	0.53	0.37
				1	0.15	0.51	
				2	0.15	0.38	

Table 5.19: Leave-One-Subject-Out results for Maximum and Minimum Angles

LEAVE ONE SUBJECT OUT							
Model	Imbalance Method	Cost Sensitive	Parameters	Class	Precision	Recall	F1 macro
AREAS							
SVM	x	✓	$kernel = rbf$ $C = 10$ $gamma = 0.001$	0	0.92	0.75	0.45
				1	0.18	0.26	
				2	0.22	0.58	
Random Forest	Random UnderSampler	x	$criterion = entropy$ $max\ depth = 80$ $max\ features = sqrt$ $min\ samples\ leaf = 1$ $min\ samples\ split = 10$ $n\ estimators = 110$	0	0.88	0.92	0.56
				1	0.33	0.21	
				2	0.49	0.53	
Decision Tree	x	✓	$criterion = entropy$ $max\ depth = 6$ $max\ features = 8$ $min\ samples\ leaf = 2$ $min\ samples\ split = 2$ $max\ leaf\ nodes = None$	0	0.88	0.63	0.37
				1	0.10	0.23	
				2	0.17	0.42	
DISTANCES							
SVM	x	✓	$kernel = linear$ $C = 10$	0	0.91	0.65	0.47
				1	0.12	0.30	
				2	0.26	0.57	
Random Forest	Random UnderSampler	x	$criterion = entropy$ $max\ depth = 10$ $max\ features = sqrt$ $min\ samples\ leaf = 1$ $min\ samples\ split = 4$ $n\ estimators = 110$	0	0.88	0.88	0.50
				1	0.24	0.15	
				2	0.33	0.56	
Decision Tree	x	✓	$criterion = entropy$ $max\ depth = 5$ $max\ features = 7$ $min\ samples\ leaf = 5$ $max\ leaf\ nodes = None$	0	0.91	0.58	0.35
				1	0.12	0.36	
				2	0.15	0.38	

Table 5.20: Leave-One-Subject-Out results for Areas and Distances

5.2 Results of Deep Learning Models

We conducted a series of experiments to evaluate the performance of the pre-trained ResNet18, the pre-trained ResNet50 and the simple CNN model on pain intensity classification.

First of all, in order to enhace the robustness of our models to variations in image appearance, as a preprocessing step, we applied several data augmentation techniques to the training images using the Albumentations library:

1. **Horizontal Flip** : The technique involves flipping the input images horizontally, which is similar to reflecting the image across a horizontal axis.
2. **Affine Transformation** : The technique involves rotating the input image between -30 and 30 degrees and randomly zooming it between 80% and 120%.
3. **Gaussian Noise** : The technique involves adding random Gaussian noise to the input images. By setting *var_limit* to (10.0, 50.0) we are indicating that the Gaussian noise will be generated using random variances between 10.0 and 50.0.
4. **Gaussian Blur** : The technique involves applying a Gaussian filter to the input images to blur them. We set the *blur_limit* parameter to 23 and we created a generator to give random values in the range from 0.1 to 2.0 to the *sigma_limit*.
5. **Color Jitter** : The technique involves applying random changes to the brightness, contrast, saturation, and hue of the input images. We set the brightness parameter to 0.2, the contrast parameter to 0.2, and the hue parameter to 0.5.
6. **Normalization** : Normalization is commonly performed on the entire training data set by subtracting the dataset's mean value from each pixel value and then dividing by the dataset's standard deviation. This ensures that the input images have identical statistical features, which can boost the model's performance. We experimented with the two most common normalization techniques:
   - **ImageNet Normalization**: The mean and standard deviation values used for ImageNet normalization are:
     Mean : [0.485, 0.456, 0.406]
     Standard deviation : [0.229, 0.224, 0.225]
   - **Zero-mean unit variance Normalization** : The mean and standard deviation values used for this normalization are:
     Mean : [0, 0, 0]
     Standard deviation : [1, 1, 1]

Data augmentation improves generalization and reduces overfitting by exposing the model to a wider range of image transformations, encouraging it to recognize underlying features rather than memorize the training data. Results in Table 5.21, showed that the data augmentation techniques that included ImageNet Normalization improved the performance of the ResNet18 model.

In our experiments, the Adam optimizer is employed. This optimizer monitors and modifies the learning rate of each network parameter individually throughout training. The learning rate for each weight is adjusted based on the predicted first and second gradients. We set the learning rate to 0.0001, which implies that the optimizer will make very tiny changes to the model's parameters during each training iteration in order to minimise the loss function. Furthermore, a weight decay of 0.000005 is employed to add an additional term to the loss function that discourages large weights. This ensures that the model's parameters are not updated too frequently during training, which can help to reduce overfitting and improve the model's performance. We also provided the MultiStepLR scheduler, which modifies the optimizer's learning rate at certain milestones. The milestones are defined as [15,23], which indicates that at the 15th and 23rd epochs, the learning rate will be reduced by a factor of 0.1. Reducing the learning rate as the model gets closer to the optimal solution, helps the model converge to a better result. All the ResNet18, ResNet50 and CNN models are trained using a batch size of 32 and for 30 epochs.

			Patient-Wise Splitting 1				
Model	Training	Augmentation	Confusion Matrix	Class	Precision	Recall	F1 macro
ResNet 18	Transfer Learning 1	Flip Imagenet Normalization	$\begin{bmatrix} 7027 & 309 & 611 \\ 879 & 61 & 125 \\ 279 & 107 & 230 \end{bmatrix}$	0	0.86	0.88	0.41
				1	0.13	0.06	
				2	0.24	0.37	
ResNet 18	Transfer Learning 1	Flip Affine Transformation Gaussian Blur Imagenet Normalization	$\begin{bmatrix} 6927 & 389 & 631 \\ 868 & 71 & 126 \\ 308 & 77 & 231 \end{bmatrix}$	0	0.85	0.87	0.41
				1	0.13	0.07	
				2	0.23	0.38	
ResNet 18	Transfer Learning 1	Flip Affine Transformation Gaussian Blur Color Jitter Imagenet Normalization	$\begin{bmatrix} 7771 & 150 & 26 \\ 980 & 83 & 2 \\ 282 & 147 & 187 \end{bmatrix}$	0	0.86	0.98	0.49
				1	0.22	0.08	
				2	0.87	0.30	
ResNet 18	Transfer Learning 1	Flip Affine Transformation Gaussian Noise Imagenet Normalization	$\begin{bmatrix} 6901 & 295 & 751 \\ 859 & 66 & 140 \\ 284 & 94 & 238 \end{bmatrix}$	0	0.86	0.87	0.41
				1	0.15	0.06	
				2	0.21	0.39	
ResNet 18	Transfer Learning 1	Flip Normalization (Mean = 0 Std = 1)	$\begin{bmatrix} 7280 & 96 & 571 \\ 893 & 51 & 121 \\ 289 & 106 & 221 \end{bmatrix}$	0	0.86	0.92	0.42
				1	0.20	0.05	
				2	0.24	0.36	
ResNet 18	Transfer Learning 1	Flip Affine Transformation Gaussian Noise Color Jitter Normalization (Mean = 0 Std = 1)	$\begin{bmatrix} 7104 & 146 & 697 \\ 866 & 61 & 138 \\ 237 & 111 & 268 \end{bmatrix}$	0	0.87	0.89	0.31
				1	0.19	0.06	
				2	0.24	0.44	

Table 5.21: Different Augmentation Combinations

The pre-trained models are adapted to our task by replacing the last FCL with a FCL with 3 output neurons to predict the three intensities. The weights from the pre-trained model are utilized as the initial weights for the new model. Three different transfer learning approaches for the pre-trained neural networks were implemented:

1. **Transfer Learning 1**: The weights of the last fully connected layer, are randomly initialized and then trained from scratch on our task while the weights of the rest of the network remain constant.
2. **Transfer Learning 2**: The entire network is re-trained.
3. **Transfer Learning 3**: The weights of some of the layers are frozen and not updated during training, while the weights of the remaining layers are randomly initialised and trained.

In addition, we evaluated the performance of several loss functions including categorical cross-entropy loss, focal loss, and soft cross-entropy loss. Cross-entropy is a frequently employed loss function in classification problems, whereas focal loss is a cross-entropy variant that addresses the issue of class imbalance. Soft cross-entropy is a smoother variation of cross-entropy that is more resistant to label noise. The output of our models is 3 scores s_1, s_2 and s_3 one for each intensity.
The softmax function is applied to assign probabilities to the scores:

$$\hat{y}_i = \frac{e^{s_i}}{\sum_{k=1}^{3} e^{s_k}} \tag{5.1}$$

,where $\hat{y}_i$ is the probability of intensity i.
Categorical Cross Entropy Loss: It is a measure of the dissimilarity between the predicted and true probability distributions. Given the predicted probabilities, the cross-entropy loss is determined as the negative log-likelihood of the true labels. It is especially effective in problems involving the classification of multiple classes since it can manage the situation in which several class labels are incompatible with one another.
The categorical cross-entropy loss for a training sample x_i is defined as:

$$L_{CE} = -\sum_{i=1}^{3} y_i \; log(\hat{y}_i) \tag{5.2}$$

,where y_i is the true probability distribution and $\hat{y}_i$ is the predicted probability distribution.
Soft Cross Entropy Loss: In contrast with the cross-entropy loss function, the SCE loss function employs a soft target distribution. This implies that the target class can have a value between 0 and 1, representing the probability of that class being the correct one, and the other classes can also have values greater than 0, indicating their probabilities as well. This is different from the traditional Cross Entropy Loss, where the target class is represented by 1 and all other classes are represented by 0. By representing the true class as a probability value between 0 and 1, the model can account for cases where the class assignment is not entirely certain.
The SCE loss can be calculated as:

$$L_{SCE} = -\sum_{i=1}^{3} (y_i \; log(\hat{y}_i) + (1 - y_i log(1 - \hat{y}_i) \tag{5.3}$$

,y_i is the soft label of the true class.
Focal Loss: While the standard cross-entropy loss function considers all samples equally, the model may give more importance to the majority of examples, and so perform poorly. In order to solve this problem, Focal Loss assigns a higher weight to the samples that are misclassified with high confidence and a lower weight to the samples that are misclassified with lower confidence. Specifically, the weight is defined as a decreasing function of the anticipated probability of the true class.
The mathematical formula of Focal Loss is:

$$L_{FC} = -\sum_{i=1}^{3} (1 - \hat{y}_i)^{\gamma} \; log(\hat{y}_i) \tag{5.4}$$

,where γ is the focusing parameter.
We applied the Transfer-Learning 2 to the ResNet models using each of these loss functions and evaluated their performance. The results in Tables 5.22 and 5.23 showed that using different loss functions did not significantly improve performance. However, cross-entropy generally performed slightly better for our dataset and task, and we, therefore, chose to use it as the final loss function for our further experiments.

			Patient-Wise Splitting 1					Sequence-Wise Spliting				
Model	Training	Augmentation	Confusion Matrix	Classes	Precision	Recall	F1 macro	Confusion Matrix	Classes	Precision	Recall	F1 macro
ResNet 50	Transfer Learning 1	Flip Imagenet Normalization	$\begin{bmatrix}6142 & 1537 & 268\\631 & 403 & 31\\372 & 34 & 210\end{bmatrix}$	0 1 2	0.86 0.20 0.41	0.77 0.38 0.34	0.48	$\begin{bmatrix}6421 & 150 & 202\\442 & 602 & 137\\204 & 125 & 314\end{bmatrix}$	0 1 2	0.91 0.27 0.48	0.79 0.51 0.49	0.56
ResNet 50	Transfer Learning 1	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}6381 & 1361 & 205\\643 & 386 & 36\\400 & 38 & 178\end{bmatrix}$	0 1 2	0.86 0.22 0.42	0.80 0.36 0.29	0.48	$\begin{bmatrix}6406 & 1551 & 166\\422 & 705 & 54\\208 & 143 & 292\end{bmatrix}$	0 1 2	0.91 0.29 0.57	0.79 0.60 0.45	0.58
ResNet 50	Transfer Learning 2	Flip Imagenet Normalization	$\begin{bmatrix}7403 & 296 & 248\\990 & 12 & 63\\328 & 15 & 273\end{bmatrix}$	0 1 2	0.85 0.04 0.47	0.93 0.01 0.44	0.45	$\begin{bmatrix}7872 & 191 & 60\\687 & 401 & 93\\232 & 85 & 326\end{bmatrix}$	0 1 2	0.90 0.59 0.68	0.97 0.34 0.51	0.65
ResNet 50	Transfer Learning 2	Flip Affine Transformation Gaussian Blur Imagenet Normalization	$\begin{bmatrix}7558 & 313 & 76\\881 & 142 & 42\\357 & 69 & 190\end{bmatrix}$	0 1 2	0.86 0.27 0.62	0.95 0.13 0.31	0.50	$\begin{bmatrix}7702 & 352 & 69\\493 & 576 & 112\\335 & 12 & 296\end{bmatrix}$	0 1 2	0.90 0.61 0.62	0.95 0.49 0.46	0.67
ResNet 50	Transfer Learning 2 + Focal Loss	Flip Affine Transformation Gaussian Noise Imagenet Normalization	$\begin{bmatrix}7419 & 289 & 239\\924 & 113 & 28\\298 & 53 & 265\end{bmatrix}$	0 1 2	0.86 0.20 0.50	0.92 0.11 0.43	0.46	$\begin{bmatrix}7288 & 732 & 103\\439 & 555 & 187\\192 & 81 & 370\end{bmatrix}$	0 1 2	0.92 0.41 0.56	0.90 0.47 0.58	0.64
ResNet 50	Transfer Learning 2 + Soft Cross Entropy Loss	Flip Affine Transformation Gaussian Noise Imagenet Normalization	$\begin{bmatrix}7290 & 431 & 226\\892 & 122 & 51\\284 & 66 & 266\end{bmatrix}$	0 1 2	0.84 0.01 0.49	0.97 0.09 0.43	0.50	$\begin{bmatrix}7815 & 246 & 62\\785 & 355 & 41\\323 & 39 & 281\end{bmatrix}$	0 1 2	0.88 0.55 0.73	0.96 0.30 0.44	0.62
ResNet 50	Transfer Learning 3 (70%)	Flip Imagenet Normalization	$\begin{bmatrix}7667 & 199 & 81\\1030 & 17 & 18\\429 & 49 & 138\end{bmatrix}$	0 1 2	0.84 0.06 0.58	0.96 0.02 0.22	0.42	$\begin{bmatrix}7904 & 26 & 17\\877 & 164 & 24\\107 & 0 & 509\end{bmatrix}$	0 1 2	0.89 0.86 0.93	0.99 0.15 0.83	0.69
ResNet 50	Transfer Learning 3 (70 %)	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}7711 & 106 & 130\\1059 & 3 & 3\\425 & 96 & 95\end{bmatrix}$	0 1 2	0.84 0.01 0.42	0.97 0.00 0.15	0.50	$\begin{bmatrix}7628 & 415 & 80\\730 & 410 & 41\\234 & 67 & 342\end{bmatrix}$	0 1 2	0.89 0.46 0.74	0.94 0.35 0.53	0.64
ResNet 50	Transfer Learning 1 + Dense Layer (512)	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}3956 & 2871 & 1120\\565 & 446 & 54\\246 & 330 & 40\end{bmatrix}$	0 1 2	0.83 0.12 0.03	0.50 0.42 0.06	0.29	$\begin{bmatrix}5531 & 1267 & 1325\\580 & 339 & 262\\604 & 23 & 16\end{bmatrix}$	0 1 2	0.82 0.21 0.01	0.68 0.29 0.02	0.33
ResNet 50	Transfer Learning 1 + Dense Layer (512) + Dense Layer (256)	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}4189 & 3363 & 395\\347 & 645 & 73\\289 & 144 & 183\end{bmatrix}$	0 1 2	0.87 0.16 0.28	0.53 0.61 0.30	0.40	$\begin{bmatrix}2794 & 1955 & 3374\\609 & 218 & 354\\432 & 114 & 97\end{bmatrix}$	0 1 2	0.73 0.10 0.03	0.34 0.18 0.15	0.21

Table 5.22: Experiments with ResNet50

			Patient-Wise Splitting 1					Sequence-Wise Spliting				
Model	Training	Augmentation	Confusion Matrix	Classes	Precision	Recall	F1 macro	Confusion Matrix	Classes	Precision	Recall	F1 macro
ResNet 18	Transfer Learning 1	Flip Imagenet Normalization	$\begin{bmatrix}7139 & 334 & 474\\ 887 & 46 & 132\\ 332 & 51 & 233\end{bmatrix}$	0 1 2	0.85 0.11 0.28	0.90 0.04 0.38	0.42	$\begin{bmatrix}6756 & 1002 & 365\\ 563 & 447 & 171\\ 140 & 116 & 387\end{bmatrix}$	0 1 2	0.91 0.29 0.42	0.83 0.38 0.60	0.56
ResNet 50	Transfer Learning 1	Flip Affine Transformation Gaussian Blur Imagenet Normalization	$\begin{bmatrix}6927 & 389 & 631\\ 868 & 71 & 126\\ 308 & 77 & 231\end{bmatrix}$	0 1 2	0.85 0.13 0.23	0.87 0.07 0.38	0.41	$\begin{bmatrix}6616 & 1046 & 461\\ 610 & 345 & 226\\ 122 & 114 & 407\end{bmatrix}$	0 1 2	0.90 0.23 0.37	0.81 0.29 0.63	0.53
ResNet 18	Transfer Learning 2	Flip Imagenet Normalization	$\begin{bmatrix}7539 & 142 & 266\\ 918 & 81 & 66\\ 393 & 52 & 171\end{bmatrix}$	0 1 2	0.85 0.29 0.34	0.95 0.08 0.28	0.44	$\begin{bmatrix}7504 & 474 & 145\\ 571 & 415 & 195\\ 110 & 97 & 436\end{bmatrix}$	0 1 2	0.92 0.42 0.56	0.92 0.35 0.68	0.64
ResNet 18	Transfer Learning 2	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}7771 & 150 & 26\\ 980 & 83 & 2\\ 282 & 147 & 187\end{bmatrix}$	0 1 2	0.86 0.22 0.87	0.98 0.08 0.30	0.49	$\begin{bmatrix}7573 & 497 & 53\\ 528 & 565 & 88\\ 189 & 89 & 365\end{bmatrix}$	0 1 2	0.91 0.49 0.72	0.93 0.48 0.57	0.68
ResNet 18	Transfer Learning 2 + Focal Loss	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}7469 & 236 & 242\\ 968 & 37 & 60\\ 212 & 36 & 368\end{bmatrix}$	0 1 2	0.86 0.12 0.55	0.94 0.03 0.60	0.51	$\begin{bmatrix}7773 & 280 & 70\\ 627 & 420 & 134\\ 273 & 16 & 354\end{bmatrix}$	0 1 2	0.90 0.59 0.63	0.96 0.36 0.55	0.65
ResNet 18	Transfer Learning 2 + Soft Cross Entropy Loss	Flip Affine Transformation Gaussian Noise Color Jitter Imagenet Normalization	$\begin{bmatrix}7449 & 192 & 306\\ 934 & 63 & 68\\ 250 & 60 & 306\end{bmatrix}$	0 1 2	0.86 0.20 0.45	0.94 0.06 0.50	0.49	$\begin{bmatrix}7735 & 315 & 73\\ 565 & 546 & 70\\ 312 & 35 & 296\end{bmatrix}$	0 1 2	0.90 0.61 0.67	0.95 0.46 0.46	0.67

Table 5.23: Experiments with ResNet18

		Patient-Wise Splitting 1					Sequence-Wise Spliting				
Model	Augmentation	Confusion Matrix	Classes	Precision	Recall	F1 macro	Confusion Matrix	Classes	Precision	Recall	F1 macro
CNN	Flip Imagenet Normalization	$\begin{bmatrix}6681 & 1012 & 254\\ 804 & 201 & 60\\ 241 & 282 & 93\end{bmatrix}$	0 1 2	0.86 0.13 0.23	0.84 0.19 0.15	0.40	$\begin{bmatrix}7212 & 671 & 240\\ 764 & 342 & 75\\ 272 & 3 & 368\end{bmatrix}$	0 1 2	0.87 0.34 0.54	0.89 0.29 0.57	0.58
CNN	Flip Affine Transformation Gaussian Blur Imagenet Normalization	$\begin{bmatrix}7286 & 601 & 60\\ 908 & 77 & 80\\ 362 & 178 & 76\end{bmatrix}$	0 1 2	0.85 0.09 0.35	0.92 0.07 0.12	0.38	$\begin{bmatrix}7352 & 567 & 204\\ 545 & 592 & 44\\ 343 & 10 & 290\end{bmatrix}$	0 1 2	0.89 0.51 0.54	0.91 0.50 0.45	0.63

Table 5.24: Experiments with CNN

5.3 Overall Results

In this section, we present our experimental results by comparing the various implementations and determining the effectiveness of each approach. Our goal is to provide a clear and concise summary of the key findings, emphasizing both the strengths and limitations of each method. The significant outcomes of our study are:

- A comparison of the results between no undersampling and the *Neighborhood Cleaning Rule* method revealed that most of the times the NCR resulted in similar overall performance or in a minimal improvement. The NCR is a method for removing misclassified instances from the majority class. However, using the NCR does not always increase the model's performance because it may remove informative samples from the majority class that are significant to the model.
- A comparison of the results between no undersampling and the *Near Miss* method, revealed that while the Near Miss improved the model's performance on the minority classes, it resulted in worse overall performance. The Near Miss undersampling method reduces class imbalance by deleting instances from the majority class that are close to examples from the minority class. It probably results in worse outcomes because the Near Miss method could have removed instances from the majority class that are useful to the model, or it could have over-reduced the majority class.
- A comparison of the results between no undersampling and the undersampling methods: *Random Undersampling* and *Cluster Centroids*, revealed that both undersampling methods improved the model's performance on the minority classes, but experiments on Patient-Wise splitting 2 resulted in a decrease in accuracy for the majority class.
- The results indicate that *cost-sensitive learning* improves the performance of the models on minority classes, but may decrease accuracy on the majority class, particularly when using Patient-Wise splitting 2. The cost-sensitive learning strategy tries to improve the performance of the minority classes by adding penalties to the loss function, but this can come at the cost of lowering the accuracy of the majority class.
- A comparison of the results between the *Patient-Wise splitting 1* (the 3-way holdout splitting method) and the *Patient-Wise splitting 2* (the cross-validation method) revealed that the cross-validation method performed better overall. It is important to achieve higher scores for minority classes (pain frames) rather than majority class, thus a lower accuracy for the majority class is acceptable. Patient-Wise splitting 2 showed higher recall scores for minority classes and was better at detecting and categorizing instances from these classes. Cross-validation is considered more robust than 3-way holdout splitting as it trains and evaluates the model on multiple folds independently, yielding a more reliable evaluation.
- A comparison of the results between the cost-sensitive learning and the under-sampling techniques such as the Random Under-sampling and Cluster Centroids revealed that on average, cost-sensitive learning had better results. Cost-sensitive learning outperforms under-sampling as it adds penalties to the loss function to better address class imbalance allowing the model to be more sensitive to the minority classes while still taking the majority class into account. Undersampling techniques, on the other hand, such as Random Undersampling and Cluster Centroids, are focusing on removing examples from the majority class in order to balance the class distribution. While this may increase performance on the minority class, it may also result in the elimination of critical majority cases.
- The experiments showed that splitting the dataset based on *Patient-Wise Splitting 1* resulted in poorer performance compared to *Sequence-Wise Splitting*. Patient-Wise splitting exposes the model to a limited set of patients and their sequences, making it harder for the model to generalize to new patients and sequences as the model memorizes specific patients rather than learning generalizable features. On the other hand, when the Sequence-Wise Splitting is used, the model is exposed to multiple sequences from the same patients, which allows it to learn generalizable features from the patients it has seen before and then test the generalization ability on unseen sequences.
- *Transfer Learning 2* overall results in better performance than *Transfer Learning 1* or *Transfer Learning 3*. This may be due to the fact that Transfer Learning 2 allows the model to alter the pre-trained weights

to better fit our specific task and dataset. It allows the model to learn more complicated features and higher-level abstractions unique to the pain expressions.

- Augmentation of the CNN input images does not clearly improve performance. One possibility is that the dataset is already varied and that expanding it further does not add any additional useful information for the model to learn.
- A comparison of the results between the CNN networks and the simple classifiers revealed that CNN models outperform simple classifiers. This may be due to the fact that CNN models can learn features automatically from input data, whereas simple classifiers rely on hand-crafted features. CNNs use a hierarchical framework to learn increasingly complicated characteristics at successive levels, allowing them to better capture the underlying patterns in the data.
- A comparison of the results between the pre-trained networks and the CNNs revealed that pre-trained models have better results than CNNs. Pre-trained models have already been trained on a huge dataset of facial images that enables them to learn features and representations specific to faces. This prior information allows them to outperform CNNs that are trained from start. Furthermore, pre-trained models have previously been fine-tuned, which means they have been altered to perform effectively on a similar task, which can also contribute to their improved performance.
- A comparison of the results between Transfer Learning 1 and Transfer Learning 1 with additional dense layers before the last dense layer revealed that adding layers decreases performance. That might have happened because more layers increase the complexity of the model and can lead to overfitting.

The best performing ResNet18, ResNet50, and CNN models were selected:

1. The ResNet18 with the Transfer Learning 2 approach, the Sequence-Wise Splitting and the augmentation techniques: Flip, Affine Transformation, Gaussian Noise, Color Jitter and Imagenet Normalization
2. The ResNet50 with the Transfer Learning 2 approach, the Sequence-Wise Splitting and the augmentation techniques: Flip, Affine Transformation, Gaussian Noise and Imagenet Normalization
3. The CNN with the Sequence-Wise Splitting and the augmentation techniques: Flip, Affine Transformation, Gaussian Blur and Imagenet Normalization

For 14 patients, we selected 2 sequences containing frames with pain. One sequence was part of the training set and the other one was part of the test set. The test sequences were not seen by the models during training and were used to evaluate the models' ability to generalize new data. Each figure in Tables 5.25 and 5.26 visualizes the true and the estimated intensities of the frames of a sequence. The figures offer a graphical depiction of how the model performed on each patient. According to the figures, in most cases, networks were able to identify painful frames in new sequences. However, there were times when the networks struggled to detect the pain frames, indicating that the models' performance can be improved.

Furthermore, we selected the model with the best overall performance using the Patient-Wise Splitting 1. Specifically, the ResNet50 with the Transfer Learning 1 approach and the augmentation techniques: Flip and Imagenet Normalization. This splitting contains 6 patients in the test set that the models haven't seen before, with all of their frames included. For a sequence of each patient, we display the actual and predicted labels for each frame. The figures in Table 5.27 demonstrate that the model is able to identify the event of pain in a new patient, even if it misses a lot of pain frames. The model has successfully learned to recognize some pain patterns rather than just memorizing specific frames, and though it may not pinpoint all painful frames accurately, it can still identify overall pain events, making it valuable for pain monitoring and management applications.

Overall, the figures demonstrate the potential of using neural networks for identifying pain events in sequences, but also highlight the need for further optimization the models.

Table 5.25: Sequences that belong to Test set of Sequence-Wise splitting

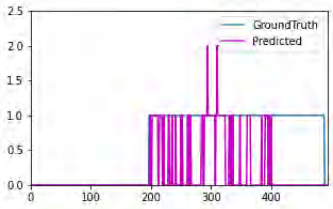
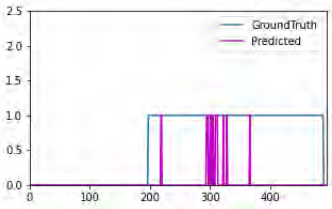
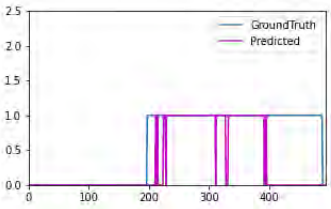
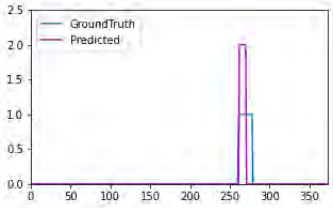
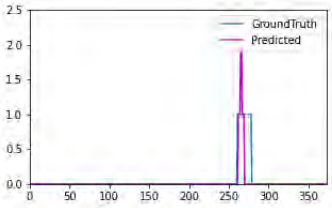
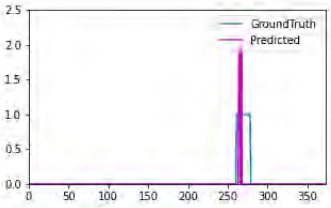
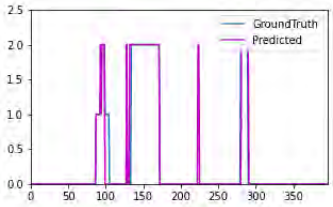
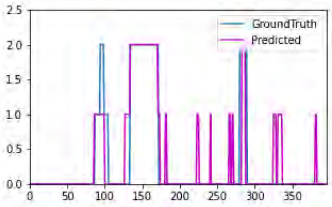
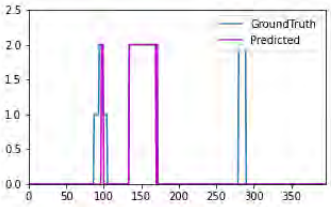
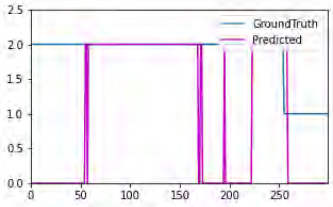
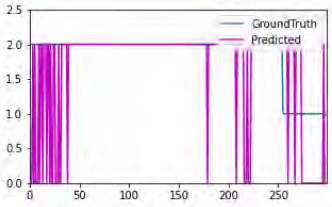
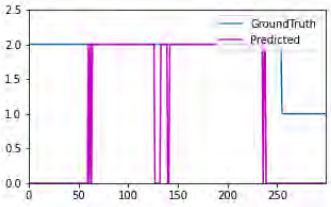
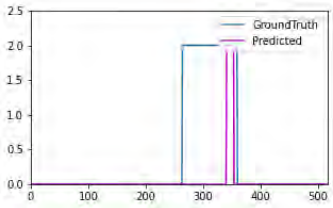
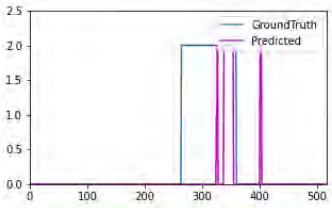
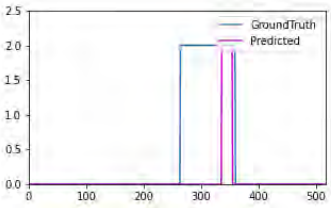
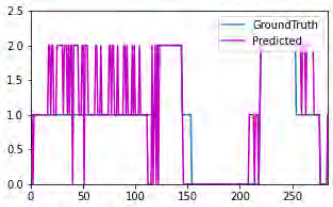
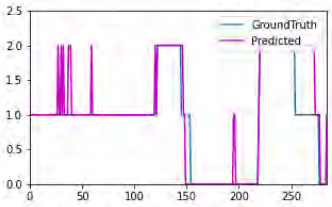
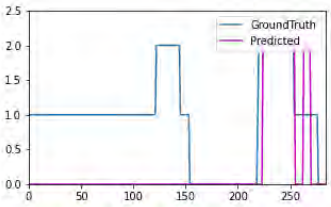
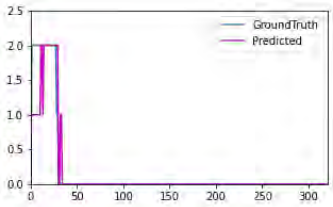
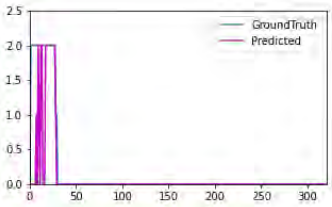
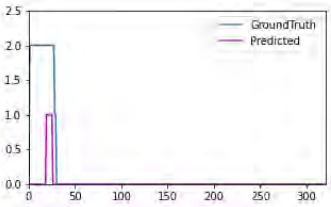
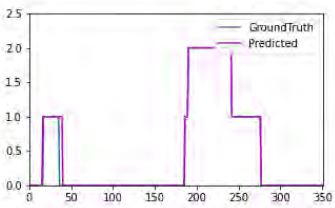
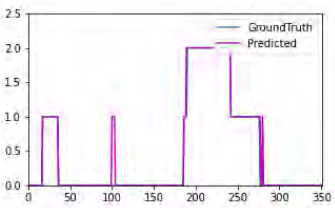
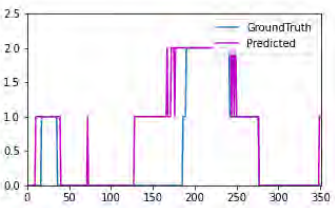
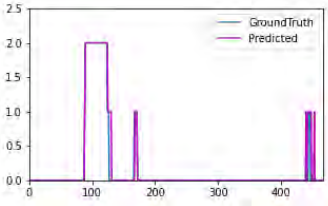
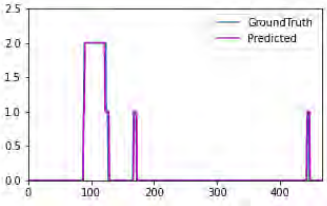
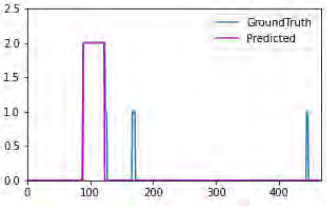
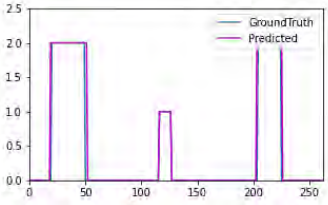
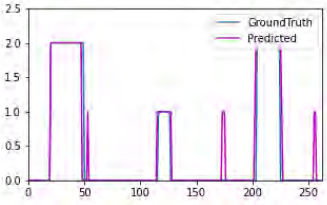
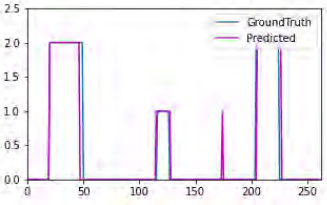
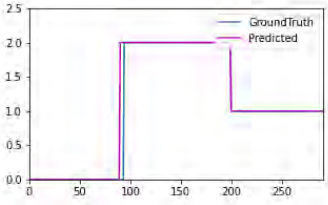
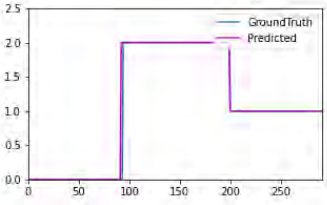
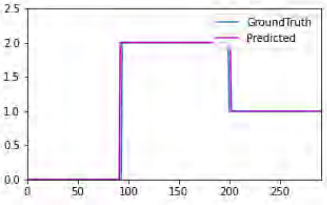
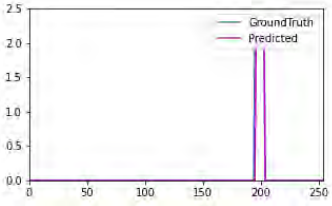
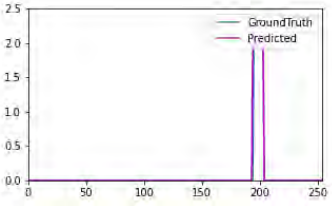
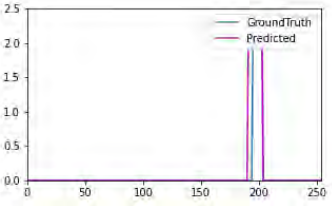
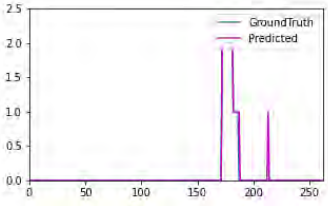
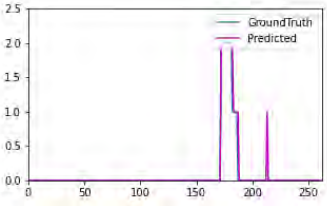
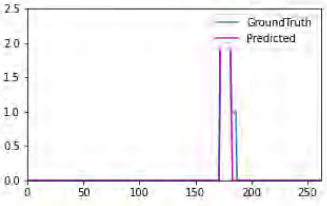
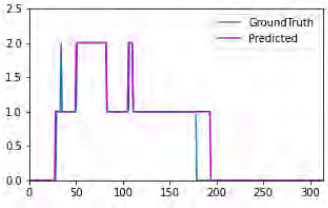
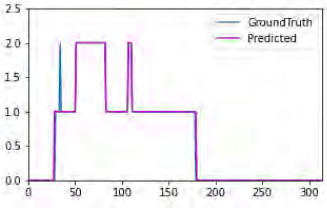
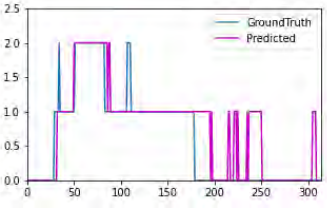
Patient	ResNet50	ResNet18	CNN
4			
5			
7			
11			
13			
17			
19			

Table 5.26: Sequences that belong to Training set of Sequence-Wise splitting

Patient	ResNet50	ResNet18	CNN
4			
5			
7			
11			
13			
17			
19			

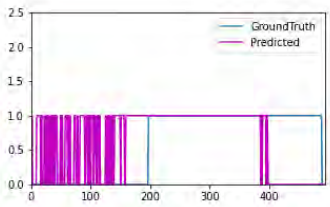
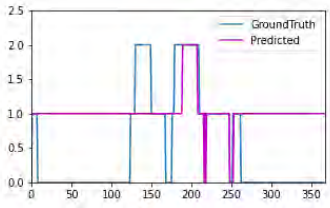
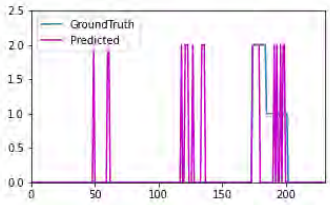
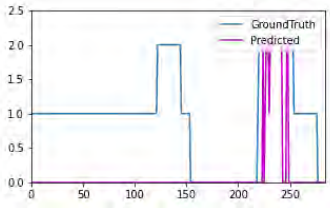
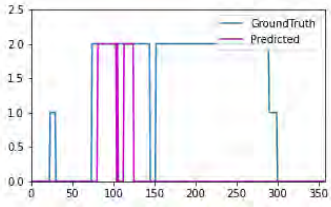
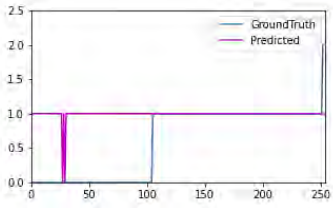
Patient	ResNet50
4	
6	
12	
17	
23	
24	

Table 5.27: Sequences that belong to Test set of Patient-Wise Splitting 1

6 Discussion and Conclusions

The aim of this study was to estimate three pain intensity levels from facial images through the implementation of two approaches. The first approach utilized simple classifiers such as SVM, RF, and DT with geometrical and appearance features to determine if changes in face geometry can predict pain intensity. The second approach applied CNNs using different transfer learning methods, including pre-trained ResNet18 and ResNet50, and a proposed CNN, to understand the extent to which CNNs can automatically estimate pain. In the deep learning approach, a preprocessing procedure was performed to ensure the faces were accurately represented in the images. This involved cropping and centralizing the faces, as well as aligning them based on the angle between the line connecting the right and left corners of the eyes and the horizontal axis. Both approaches in this study addressed the challenge of imbalanced data. In the first approach, different methods such as under-sampling and cost-sensitive learning were used, while in the second approach, a Weighted Random Sampler was employed along with various data augmentation techniques. Using the UNBC dataset, the study took into consideration the challenges and limitations of the dataset to develop an unbiased estimator that could generalize to new patients. The findings and contributions of the proposed methods are summarized and the limitations of automatic pain assessment are discussed, along with potential avenues for future research.

Despite our diligent efforts to attain optimal results, our study did not yield the anticipated outcome due to a confluence of several factors. To begin with, the UNBC-McMaster Pain Archive database has a limited sample size of patients, which affects the data's representativeness. In general, small sample size can lower the statistical power of a study as a model is not trained in a diverse range of patients. The procedure of identifying underlying patterns associated with the classification objective becomes more complex, resulting in a model that may be unable to generalize effectively to a larger population.

Additionally, the fact that the UNBC-McMaster Pain Archive database is an imbalanced dataset, introduces greater difficulty in developing a model which can properly capture the characteristics of the underrepresented minority classes. Therefore, it is important to consider an optimal splitting strategy to ensure the validity of the results. Many works in the literature, rely on a simple random splitting of frames or they balance the dataset at the beginning of their analysis, leading to seemingly good results. However, these results are successful for the classification of the specific dataset and do not guarantee the ability of the model to generalize to new patients. As a matter of fact, balancing the dataset to improve the classification of images without considering the impact on the generalizability of the model may result in an artificial distribution of the classes, which does not reflect the true distribution in real-world problems. The model may be more prone to overfitting, as it may learn the specific class distribution in the balanced dataset, rather than the general patterns in the data.

Contrary to these approaches, we propose two splitting strategies based on stratification, which ensures that each set has a representative sample of instances from each class. The first method involves patient-wise splitting. Patient-wise splitting is often considered a better approach than random splitting in medical applications because it helps to ensure that the training and test sets are independent and that the model is not trained on and evaluated on data from the same patients which can lead to overfitting due to the specific characteristics of one particular patient. The second splitting is based on sequence-wise splitting. Sequences from the same individual can be included in both the training and testing sets, but frames from the same sequence cannot. In medical applications, sequence-wise splitting is preferred over random splitting for pain analysis as it considers temporal relationships between frames. Pain can change over time in terms of

intensity, duration, or frequency. Sequence-wise splitting provides a more accurate representation of the data preserving these relationships. This allows models to better capture the changing nature of pain and make more accurate predictions. We examined the effectiveness of both patient-wise and sequence-wise splitting and our findings show that, although the improvement may be small, sequence-wise splitting has a better performance. This may be due to the fact that patient-wise splitting may result in a biased representation of the data, as the model may not be exposed to the full range of variability in the data. Based on the literature and our results the problem of pain intensity classification can be approached in two different ways: generalizing to different sequences of the same patient or generalizing to unique patients.

Another crucial point is the effectiveness of stratified splitting of the dataset into training, test, and validation sets, followed by under-sampling in the training set compared to the initial balancing of the dataset. In theory, balancing the set from the beginning contributes to the creation of balanced training and validation sets. A balanced validation set may not accurately reflect the real-world distribution of the data, leading to over-optimistic performance evaluations. On the other hand, having balanced training to ensure that the model is trained on a representative sample of the data and a validation set preserving the same distribution as the real-world data provide a more reliable assessment of the model's performance in real-world scenarios. However, based on our findings we observe that only balancing the training set can also lead to over-fitting, particularly if the training set is small. Furthermore, while under-sampling on the training set has been shown to increase minority class prediction, it can also result in a decrease in performance for the majority class. Under-sampling the majority class reduces the amount of training data for the model, leading to a loss of information that negatively impacts the model's ability to accurately predict the majority class.

Stratifying the dataset and cost-sensitive learning addresses the issue of class imbalance in a more sophisticated way than balancing the initial dataset or the training set. Cost-sensitive learning adjusts the loss function to take into account the costs of false predictions, providing the model with a more nuanced understanding of the importance of correctly classifying samples from the minority classes. Cost-sensitive learning allows the model to learn from all samples rather than removing any data.

Our results cannot be directly compared to the literature as the best models in the literature are obtained using different criteria and pre-processing techniques, such as reducing the number of frames, which affects the performance of the model. After selecting a model using alternative validation methods, such as leave-one-patient-out cross-validation could be appropriate for final model evaluation.

In the results section, classifiers using hand-crafted features, such as geometric features and HOGs, were demonstrated to underperform in predicting pain for a variety of reasons. The accuracy of the pain estimation depends on the quality and robustness of the features used. Pain is a subjective experience that can vary greatly from person to person and can be affected by various factors such as age, gender, mental state, and culture. As a result, hand-crafted features may not be able to accurately capture this subjectivity. Additionally, there is significant inter-individual variability in physical features, and hand-crafted features may not be able to account for these differences, such as the difference in the appearance of the same facial expression on different individuals. Furthermore, hand-crafted features may not be expressive enough to capture the nuanced and complex expressions of pain, such as the difference in pain intensity conveyed by the same facial expression. The results indicated that Random forests have better performance for pain estimation compared to SVMs and Decision trees. This approach enhances the accuracy of the predictions by capturing the variability in the data. In addition, random forests are better equipped to handle such imbalanced data. The technique also provides feature importance scores for each feature in the dataset, allowing practitioners to identify the most important features for pain estimation and improve the model through feature selection. Additionally, random forests reduce the risk of overfitting, a common problem in decision trees where the model learns the noise in the data instead of the true pattern, by combining multiple decision trees and averaging their predictions. At the same time, it seems that SVMs have several advantages over Decision trees when it comes to pain intensity estimation. Firstly, SVMs are considered more flexible than decision trees, as they can model both linear and non-linear relationships between features and the target variable, whereas decision trees are limited to non-linear modeling. Additionally, SVMs are better at handling high-dimensional data, as they use the kernel trick to project data into a higher dimensional space, whereas decision trees can become less effective as the number of features increases. Lastly, SVMs are

considered to be more robust to outliers and noisy data, as they use a margin to separate the classes, while decision trees, which are based on a greedy approach that splits the data based on the feature that provides the largest information gain, are less robust.

Our findings reveal that CNNs outperform simple classifiers with hand-crafted features. The use of CNNs allows automatic feature extraction, potentially resulting in a better representation of complex pain patterns in images. Furthermore, pre-trained CNNs that have been trained on facial tasks generally have better performance compared to those that have been trained on other tasks, such as ImageNet. This is because pre-trained CNNs trained on facial tasks are better suited for facial-related tasks, such as pain estimation, as they have already learned to identify and extract facial features and patterns relevant to the task. Another reason for the better performance is the use of transfer learning, where the model learns from both the pre-trained features and the new task-specific features. Pre-trained CNNs can be fine-tuned to the specific task of pain estimation by training on a smaller dataset of pain images. This leverages the knowledge learned from the pre-training task, allowing the model to adapt to the new task with less data, speeding up the training process, and improving performance. In contrast, pre-trained CNNs that are trained on ImageNet have learned to extract features from a wider range of images, which may not be relevant to the task of pain estimation. In addition, the cross-entropy loss does not take into account the magnitude of the error: It only focuses on the difference between the predicted probability and the true label, without considering the magnitude of the error. In pain assessment, the magnitude of the error can be important, as a small error in prediction can have a big impact on the patient's pain management. It does not take into account the ordinal nature of the pain scale. The pain scale is an ordinal scale, meaning that the difference between the levels of pain is not equal. Furthermore, Cross-entropy loss does not take into account the spatial relationship and the continuity between the facial features.

Automated pain detection systems exhibit limitations in the accuracy of their predictions due to the challenges associated with collecting and utilizing high-quality, relevant data. Firstly, people have different pain tolerance levels, meaning that what one person considers severe pain, another person may view it as mild. This variability can make it difficult for an automatic system to accurately identify the presence of pain, especially if individuals try to hide or suppress their pain expressions. The available datasets are not representative of the population, which can lead to bias towards certain groups. The reaction to pain can vary greatly among premature babies, neonates, children, teenagers, and adults, making it challenging to build a unified model that covers all categories. Additionally, wearable objects such as spectacles or sunglasses or hair may partially cover the face, deteriorating the quality of extracted features, thus challenging the models. People from different cultures and backgrounds may express and understand pain differently, which is an important consideration when using facial analysis models for pain assessment. The psychological state of patients can also affect their behaviors as emotions such as anxiety, depression, stress, and cognitive factors like attention and expectations can influence how an individual perceives and reports their pain. There are also various sources of variability, such as changes in lighting conditions, head movements, and changes in facial expressions over time. The dataset used in the current work only focused on pain caused by shoulder injuries in adults, but in reality, pain can be caused by various factors affecting facial expressions, body movements, and body language.

Despite the limitations and challenges, our study provides valuable insights into the subject and serves as a foundation for future research to build upon. In the field of pain assessment, there are several potential avenues for future work that can lead to improved accuracy and practical applications. Using feature selection methods in conjunction with pain assessment algorithms could help select the most relevant features for estimating pain intensity which can lead to improved performance and reduced complexity. In addition, in present studies, pain intensity is estimated based on a single modality, such as facial images. However, combining information from multiple modalities, such as facial images, physiological signals, and self-reported pain, could provide a more comprehensive understanding of a person's pain experience. There are several fusion techniques that can be used to combine information from multiple modalities, including early fusion, late fusion, and hybrid fusion. By exploring these or other fusion techniques, future work in the field of pain assessment could lead to more accurate and reliable pain intensity estimates. Additionally, incorporating fusion methods into pain assessment algorithms could help overcome some of the limitations of single-modality approaches, such as decreased accuracy in certain populations or conditions. Another promising avenue could be the incorporation of temporal information. Most pain intensity estimation algorithms consider a single

snapshot of a person's facial expression, rather than considering changes over time. However, incorporating temporal information could provide a better understanding of the dynamics of pain intensity. Additionally, considering the sequence of pain intensity estimates could provide valuable insights into the evolution of a person's pain experience. There are several approaches that can be used to incorporate temporal information including recurrent neural networks (RNNs), convolutional LSTMs (ConvLSTMs), and gated recurrent units (GRUs). To increase the confidence in automatic pain assessment used in a clinical setting, further studies should be conducted to assess how it varies between subjects. Collecting data from a more diverse population, with individuals experiencing different types of pain with variations in the intensity, duration, and location of pain and under different conditions could lead to more generalizable algorithms that are robust of pain intensity changes.

Bibliography

[1] Laocoön and his sons, Oct 2020.

[2] Y-I Tian, Takeo Kanade, and Jeffrey F Cohn. Recognizing action units for facial expression analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 23(2):97–115, 2001.

[3] Kenneth M Prkachin. The consistency of facial expressions of pain: a comparison across modalities. *Pain*, 51(3):297–306, 1992.

[4] Mariette Awad and Rahul Khanna. *Efficient learning machines: theories, concepts, and applications for engineers and system designers*. Springer nature, 2015.

[5] John Masapanta. Machine and deep learning for lithofacies classification from well logs in the north sea. Master's thesis, uis, 2021.

[6] Lin Hao, Juncheol Kim, Sookhee Kwon, and Il Do Ha. Deep learning-based survival analysis for high-dimensional survival data. *Mathematics*, 9(11):1244, 2021.

[7] Van Hiep Phung and Eun Joo Rhee. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, 9(21):4500, 2019.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Delaunay triangulations: height interpolation. *Computational geometry: algorithms and applications*, pages 191–218, 2008.

[10] Jonathan Richard Shewchuk. Lecture notes on delaunay mesh generation. 1999.

[11] Yuxing Chen, Peter Goetsch, Mohammad A Hoque, Jiaheng Lu, and Sasu Tarkoma. d d-simplexed: Adaptive delaunay triangulation for performance modeling and prediction on big data analytics. *IEEE Transactions on Big Data*, 8(2):458–469, 2019.

[12] Ioannis Markoulidakis, Ioannis Rallis, Ioannis Georgoulas, George Kopsiaftis, Anastasios Doulamis, and Nikolaos Doulamis. Multiclass confusion matrix reduction method and its application on net promoter score classification problem. *Technologies*, 9(4):81, 2021.

[13] Ashish Semwal and Narendra D Londhe. Automated pain severity detection using convolutional neural network. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 66–70. IEEE, 2018.

[14] Patrick Lucey, Jeffrey F Cohn, Kenneth M Prkachin, Patricia E Solomon, and Iain Matthews. Painful data: The unbc-mcmaster shoulder pain expression archive database. In *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*, pages 57–64. IEEE, 2011.

[15] Lucas PJJ Noldus, Fabrizio Grieco, Leanne WS Loijens, and Patrick H Zimmerman. Measuring behavior 2005. In *5th International Conference on Methods and Techniques in Behavioral Research, Wageningen, The Netherlands*, 2005.

[16] Charles Darwin and Phillip Prodger. *The expression of the emotions in man and animals*. Oxford University Press, USA, 1998.

[17] Eeva A Elliott and Arthur M Jacobs. Facial expressions, emotions, and sign languages. *Frontiers in psychology*, 4:115, 2013.

[18] P Ekmann. Universal facial expressions in emotion. *Studia Psychologica*, 15(2):140, 1973.

[19] Mohammad Tavakolian. *Efficient spatiotemporal representation learning for pain intensity estimation from facial expressions.* PhD thesis, University of Oulu, Finland, 2021.

[20] Robert L Merrill. Central mechanisms of orofacial pain. *Dental Clinics*, 51(1):45–59, 2007.

[21] Ilham Seladji. Automatic pain assessment through facial expressions. 2020.

[22] Thomas Hadjistavropoulos, Kenneth D Craig, and Shannon Fuchs-Lacelle. Social influences and the communication of pain. 2004.

[23] Kenneth M Prkachin and Kenneth D Craig. Expressing pain: The communication and interpretation of facial pain signals. *Journal of nonverbal behavior*, 19(4):191–205, 1995.

[24] Howard Leventhal and Elizabeth Sharp. Facial expressions as indicators of distress. *Affect, Cognition and Personality: empirical Studies. Springer, New York*, pages 296–318, 1965.

[25] Jerry D Boucher. Facial displays of fear, sadness and pain. *Perceptual and Motor Skills*, 28(1):239–242, 1969.

[26] Kenneth M Prkachin, Neil A Currie, and Kenneth D Craig. Judging nonverbal expressions of pain. *Canadian Journal of Behavioural Science/Revue canadienne des sciences du comportement*, 15(4):409, 1983.

[27] Kenneth D Craig. The facial expression of pain better than a thousand words? *APS Journal*, 1(3):153–162, 1992.

[28] Jean-Francois Payen, Olivier Bru, Jean-Luc Bosson, Anna Lagrasta, Eric Novel, Isabelle Deschaux, Pierre Lavagne, and Claude Jacquot. Assessing pain in critically ill sedated patients by using a behavioral pain scale. *Critical care medicine*, 29(12):2258–2263, 2001.

[29] Paolo L Manfredi, Brenda Breuer, Diane E Meier, and Leslie Libow. Pain assessment in elderly patients with severe dementia. *Journal of Pain and Symptom Management*, 25(1):48–52, 2003.

[30] Kathleen A Puntillo, Ann B Morris, Carol L Thompson, Julie Stanik-Hutt, Cheri A White, and Lorie R Wild. Pain behaviors observed during six common procedures: results from thunder project ii. *Critical care medicine*, 32(2):421–427, 2004.

[31] Kenneth M Prkachin. Assessing pain by facial expression: facial expression as nexus. *Pain Research and Management*, 14(1):53–58, 2009.

[32] Kenneth D Craig, Susan A Hyde, and Christopher J Patrick. Genuine, suppressed and faked facial behavior during exacerbation of chronic low back pain. *Pain*, 46(2):161–171, 1991.

[33] Judith H Watt-Watson, Cathy Evernden, and C Lawson. Parents' perceptions of their child's acute pain experience. *Journal of Pediatric Nursing*, 5(5):344–349, 1990.

[34] Belinda Goodenough, Louise Addicoat, G David Champion, Marita McInerney, Bev Young, Kate Juniper, and John B Ziegler. Pain in 4-to 6-year-old children receiving intramuscular injections: a comparison of the faces pain scale with other self-report and behavioral measures. *The Clinical journal of pain*, 13(1):60–73, 1997.

[35] Diane L LaChapelle, Thomas Hadjistavropoulos, and Kenneth D Craig. Pain measurement in persons with intellectual disabilities. *The Clinical Journal of Pain*, 15(1):13–23, 1999.

[36] Christine T Chambers, Graham J Reid, Kenneth D Craig, Patrick J McGrath, and G Allen Finley. Agreement between child and parent reports of pain. *The Clinical journal of pain*, 14(4):336–342, 1998.

[37] Paul Ekman and Wallace V Friesen. Manual for the facial action coding system. palo alto, 1978.

[38] Emily B Prince, Katherine B Martin, Daniel S Messinger, and M Allen. Facial action coding system, 2015.

[39] P Ekman, WV Friesen, and JC Hager. Facial action coding system. the manual. salt lake city. *Network Information Research Corporation*, 2002.

[40] Yang Zhang. *Intelligent emotion recognition from facial and whole-body expressions using adaptive ensemble models*. University of Northumbria at Newcastle (United Kingdom), 2015.

[41] Bana Leelavathi. Facial expression recognition based scoring system for restaurants by using deep learning concepts. *intelligence*, 23(2):97–115, 2001.

[42] Rui Tang. *Versatile Multidimensional Pattern Analysis for Automated Facial Modeling and Architecture Parsing*. PhD thesis, University of Bath, 2017.

[43] Michel François Valstar. Timing is everything: A spatio-temporal approach to the analysis of facial actions. 2008.

[44] Xijian Fan. *Spatio-temporal Framework on Facial Expression Recognition*. PhD thesis, University of Warwick, 2016.

[45] Brais Martinez, Michel F Valstar, Bihan Jiang, and Maja Pantic. Automatic analysis of facial actions: A survey. *IEEE transactions on affective computing*, 10(3):325–347, 2017.

[46] Linda LeResche. Facial expression in pain: A study of candid photographs. *Journal of Nonverbal Behavior*, 7(1):46–56, 1982.

[47] Kenneth D Craig and Christopher J Patrick. Facial expression during induced pain. *Journal of personality and social psychology*, 48(4):1080, 1985.

[48] Miriam Kunz, Nicole Faltermeier, and Stefan Lautenbacher. Impact of visual learning on facial expressions of physical distress: A study on voluntary and evoked expressions of pain in congenitally blind and sighted individuals. *Biological psychology*, 89(2):467–476, 2012.

[49] Kenneth M Prkachin and Patricia E Solomon. The structure, reliability and validity of pain expression: Evidence from patients with shoulder pain. *Pain*, 139(2):267–274, 2008.

[50] Marc W Heft, Richard H Gracely, Ronald Dubner, and Patricia A McGrath. A validation model for verbal descriptor scaling of human clinical pain. *Pain*, 9(3):363–373, 1980.

[51] Kenneth M Prkachin, Patricia E Solomon, and Joan Ross. Underestimation of pain by health-care providers: towards a model of the process of inferring pain in others. *Canadian Journal of Nursing Research Archive*, pages 88–106, 2007.

[52] Tim Seers, Sheena Derry, Kate Seers, and R Andrew Moore. Professionals underestimate patients' pain: a comprehensive review. *Pain*, 159(5):811–818, 2018.

[53] Philipp Werner, Ayoub Al-Hamadi, and Robert Niese. Comparative learning applied to intensity rating of facial expressions of pain. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(05):1451008, 2014.

[54] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[55] Mohammadreza Sheykhmousa, Masoud Mahdianpari, Hamid Ghanbari, Fariba Mohammadimanesh, Pedram Ghamisi, and Saeid Homayouni. Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:6308–6325, 2020.

[56] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[57] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[58] Mohamed Chouaib Naimi, Mohamed Amine Naimi, and Mohammed Tewfik Benatallah. *Medical Image Classification with Convolutional Neural Network*. PhD thesis, kasdi Merbah University of OUARGLA.

[59] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9:611–629, 2018.

[60] Muhammad Waleed Zafar. *Object Detection and Segmentation using Region-based Deep Learning Architectures*. PhD thesis, Master Thesis, Technische Universität Dortmund, 2018.

[61] Laura Kölbl. *Deep Convolution Neural Networks for Image Analysis*. PhD thesis, 2017.

[62] Samsung AI Cambridge. Pre-training strategies and datasets for facial representation learning.

[63] Hang Du, Hailin Shi, Dan Zeng, Xiao-Ping Zhang, and Tao Mei. The elements of end-to-end deep face recognition: A survey of recent advances. *ACM Computing Surveys (CSUR)*, 54(10s):1–42, 2022.

[64] Zheng Sun, Andrew W Sumsion, Shad A Torrie, and Dah-Jye Lee. Learning facial motion representation with a lightweight encoder for identity verification. *Electronics*, 11(13):1946, 2022.

[65] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015.

[66] Ioannis Pikoulis, Panagiotis P Filntisis, and Petros Maragos. Leveraging semantic scene characteristics and multi-stream convolutional architectures in a contextual approach for video-based visual emotion recognition in the wild. In *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*, pages 01–08. IEEE, 2021.

[67] Mengying Shu. Deep learning for image classification on very small datasets using transfer learning. *Creat. Components, Jan*, 2019.

[68] Grega Vrbančič and Vili Podgorelec. Transfer learning with adaptive fine-tuning. *IEEE Access*, 8:196197–196211, 2020.

[69] Jacob M Williams. Deep learning and transfer learning in the classification of eeg signals. 2017.

[70] Sathiesh Kumar Kaliyugarasan. Deep transfer learning in medical imaging. Master's thesis, The University of Bergen, 2019.

[71] Nampally Tejasri and Mongkol Ekapanyapong. Material recognition using deep learning techniques. 2019.

[72] Ahmed Bilal Ashraf, Simon Lucey, Jeffrey F Cohn, Tsuhan Chen, Zara Ambadar, Ken Prkachin, Patty Solomon, and Barry J Theobald. The painful face: Pain expression recognition using active appearance models. In *Proceedings of the 9th international conference on Multimodal interfaces*, pages 9–14, 2007.

[73] Patrick Lucey, Jeffrey Cohn, Simon Lucey, Iain Matthews, Sridha Sridharan, and Kenneth M Prkachin. Automatically detecting pain using facial actions. In *2009 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops*, pages 1–8. IEEE, 2009.

[74] Rizwan Ahmed Khan, Alexandre Meyer, Hubert Konik, and Saida Bouakaz. Pain detection through shape and appearance features. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2013.

[75] Henrik Pedersen. Learning appearance features for pain detection using the unbc-mcmaster shoulder pain expression archive database. In *International Conference on Computer Vision Systems*, pages 128–136. Springer, 2015.

[76] Ruijing Yang, Shujun Tong, Miguel Bordallo, Elhocine Boutellaa, Jinye Peng, Xiaoyi Feng, and Abdenour Hadid. On pain assessment from facial videos using spatio-temporal local descriptors. In *2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6. IEEE, 2016.

[77] Moses Rupenga and Hima B Vadapalli. Automatic spontaneous pain recognition using supervised classification learning algorithms. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–6. IEEE, 2016.

[78] Junkai Chen, Zheru Chi, and Hong Fu. A new framework with multiple tasks for detecting and locating pain events in video. *Computer Vision and Image Understanding*, 155:113–123, 2017.

[79] Zakia Hammal and Jeffrey F Cohn. Automatic detection of pain intensity. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 47–52, 2012.

[80] Sebastian Kaltwang, Ognjen Rudovic, and Maja Pantic. Continuous pain intensity estimation from facial expressions. In *International Symposium on Visual Computing*, pages 368–377. Springer, 2012.

[81] Ognjen Rudovic, Vladimir Pavlovic, and Maja Pantic. Automatic pain intensity estimation with heteroscedastic conditional ordinal random fields. In *International Symposium on Visual Computing*, pages 234–243. Springer, 2013.

[82] Zuhair Zafar and Nadeem Ahmad Khan. Pain intensity evaluation through facial action units. In *2014 22nd International Conference on Pattern Recognition*, pages 4696–4701. IEEE, 2014.

[83] Ramin Irani, Kamal Nasrollahi, and Thomas B Moeslund. Pain recognition using spatiotemporal oriented energy of facial muscles. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 80–87, 2015.

[84] Neeru Rathee and Dinesh Ganotra. A novel approach for pain intensity detection based on facial feature deformations. *Journal of Visual Communication and Image Representation*, 33:247–254, 2015.

[85] Neeru Rathee and Dinesh Ganotra. Multiview distance metric learning on facial feature descriptors for automatic pain intensity detection. *Computer Vision and Image Understanding*, 147:77–86, 2016.

[86] Sourav Dey Roy, Mrinal Kanti Bhowmik, Priya Saha, and Anjan Kumar Ghosh. An approach for automatic pain detection through facial expression. *Procedia Computer Science*, 84:99–106, 2016.

[87] Rui Zhao, Quan Gan, Shangfei Wang, and Qiang Ji. Facial expression intensity estimation using ordinal information. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3466–3474, 2016.

[88] Warren Adam Shier and S Yanushkevich. Pain recognition and intensity classification using facial expressions. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3578–3583. IEEE, 2016.

[89] Mohammad Tavakolian and Abdenour Hadid. Deep binary representation of facial expressions: A novel framework for automatic pain intensity recognition. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1952–1956. IEEE, 2018.

[90] Ghazal Bargshady, Jeffrey Soar, Xujuan Zhou, Ravinesh C Deo, Frank Whittaker, and Hua Wang. A joint deep neural network model for pain recognition from face. In *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pages 52–56. IEEE, 2019.

[91] Ghazal Bargshady, Xujuan Zhou, Ravinesh C Deo, Jeffrey Soar, Frank Whittaker, and Hua Wang. Enhanced deep learning algorithm development to detect pain intensity from facial expression images. *Expert Systems with Applications*, 149:113305, 2020.

[92] Mohammad Tavakolian, Miguel Bordallo Lopez, and Li Liu. Self-supervised pain intensity estimation from facial videos via statistical spatiotemporal distillation. *Pattern Recognition Letters*, 140:26–33, 2020.

[93] Ioannis Karamitsos, IIham Seladji, and Sanjay Modak. A modified cnn network for automatic pain identification using facial expressions. *Journal of Software Engineering and Applications*, 14(8):400–417, 2021.

[94] Safaa El Morabit, Atika Rivenq, Mohammed-En-nadhir Zighem, Abdenour Hadid, Abdeldjalil Ouahabi, and Abdelmalik Taleb-Ahmed. Automatic pain estimation from facial expressions: a comparative analysis using off-the-shelf cnn architectures. *Electronics*, 10(16):1926, 2021.

[95] Yibo Huang, Linbo Qing, Shengyu Xu, Lu Wang, and Yonghong Peng. Hybnet: a hybrid network structure for pain intensity estimation. *The Visual Computer*, 38(3):871–882, 2022.

[96] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.

[97] Iain Matthews and Simon Baker. Active appearance models revisited. *International journal of computer vision*, 60(2):135–164, 2004.

[98] Simon Lucey, Rajitha Navarathna, Ahmed Bilal Ashraf, and Sridha Sridharan. Fourier lucas-kanade algorithm. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1383–1396, 2012.

[99] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[100] Siu-Wing Cheng, Tamal Krishna Dey, Jonathan Shewchuk, and Sartaj Sahni. *Delaunay mesh generation*. CRC Press Boca Raton, 2013.

[101] Jean Gallier. Notes on convex sets, polytopes, polyhedra, combinatorial topology, voronoi diagrams and delaunay triangulations. *arXiv preprint arXiv:0805.0292*, 2008.

[102] Simena Dinas and José María Banon. A review on delaunay triangulation with application on computer vision. *Int. J. Comput. Sci. Eng*, 3:9–18, 2014.

[103] Anna Bengtsson. *When mathematics teachers focus discussions on slope: Swedish upper secondary teachers in a professional development initiative*. PhD thesis, Faculty of Technology, Linnaeus university, 2014.

[104] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[105] Asim Jan. *Deep learning based facial expression recognition and its applications*. PhD thesis, Brunel University London, 2017.

[106] Delwende Eliane Birba. A comparative study of data splitting algorithms for machine learning model selection, 2020.

[107] Yun Xu and Royston Goodacre. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of analysis and testing*, 2(3):249–262, 2018.

[108] Peter de Boves Harrington. Multiple versus single set validation of multivariate models to avoid mistakes. *Critical reviews in analytical chemistry*, 48(1):33–46, 2018.

[109] Johan A Westerhuis, Huub CJ Hoefsloot, Suzanne Smit, Daniel J Vis, Age K Smilde, Ewoud JJ van Velzen, John PM van Duijnhoven, and Ferdi A van Dorsten. Assessment of plsda cross validation. *Metabolomics*, 4(1):81–89, 2008.

[110] Michael Esterman, Benjamin J Tamber-Rosenau, Yu-Chin Chiu, and Steven Yantis. Avoiding non-independence in fmri data analysis: leave one subject out. *Neuroimage*, 50(2):572–576, 2010.

[111] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[112] Pere Marti-Puig, Chiara Capra, Daniel Vega, Laia Llunas, and Jordi Solé-Casals. A machine learning approach for predicting non-suicidal self-injury in young adults. *Sensors*, 22(13):4790, 2022.

[113] Douglas M Hawkins, Subhash C Basak, and Denise Mills. Assessing model fit by cross-validation. *Journal of chemical information and computer sciences*, 43(2):579–586, 2003.

[114] Ilias Tougui, Abdelilah Jilbab, and Jamal El Mhamdi. Impact of the choice of cross-validation techniques on the results of machine learning-based diagnostic applications. *Healthcare informatics research*, 27(3):189–199, 2021.

[115] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS international transactions on computer science and engineering*, 30(1):25–36, 2006.

[116] Enislay Ramentol, Yailé Caballero, Rafael Bello, and Francisco Herrera. Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory. *Knowledge and information systems*, 33(2):245–265, 2012.

[117] Mirza Amaad Ul Haq Tahir, Sohail Asghar, Awais Manzoor, and Muhammad Asim Noor. A classification model for class imbalance dataset using genetic programming. *IEEE Access*, 7:71013–71037, 2019.

[118] Vitor Miguel Saraiva Esteves. Techniques to deal with imbalanced data in multi-class problems: A review of existing methods. 2020.

[119] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[120] Daying Quan, Wei Feng, Gabriel Dauphin, Xiaofeng Wang, Wenjiang Huang, and Mengdao Xing. A novel double ensemble algorithm for the classification of multi-class imbalanced hyperspectral data. *Remote Sensing*, 14(15):3765, 2022.

[121] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.

[122] Hakan Altınçay and Cem Ergün. Clustering based under-sampling for improving speaker verification decisions using adaboost. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 698–706. Springer, 2004.

[123] Lior Rokach and Oded Maimon. Supervised learning. In *Data Mining and Knowledge Discovery Handbook*, pages 133–147. Springer, 2009.

[124] M Mostafizur Rahman and D Davis. Cluster based under-sampling for unbalanced cardiovascular data. In *Proceedings of the world congress on engineering*, volume 3, pages 3–5, 2013.

[125] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu, and Jing-Shang Jhang. Clustering-based undersampling in class-imbalanced data. *Information Sciences*, 409:17–26, 2017.

[126] Yong-Seok Jeon and Dong-Joon Lim. Psu: Particle stacking undersampling method for highly imbalanced big data. *IEEE Access*, 8:131920–131927, 2020.

[127] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Conference on artificial intelligence in medicine in Europe*, pages 63–66. Springer, 2001.

[128] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[129] Nutthaporn Junsomboon and Tanasanee Phienthrakul. Combining over-sampling and under-sampling techniques for imbalance dataset. In *Proceedings of the 9th international conference on machine learning and computing*, pages 243–247, 2017.

[130] Małgorzata Janicka, Mateusz Lango, and Jerzy Stefanowski. Using information on class interrelations to improve classification of multiclass imbalanced data: a new resampling algorithm. *International Journal of Applied Mathematics and Computer Science*, 29(4), 2019.

[131] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. *Advances in neural information processing systems*, 10, 1997.

[132] Nilsson NJ. Learning machines: Foundations of trainable pattern-classifying systems, 1965.

[133] Fares Grina, Zied Elouedi, and Eric Lefevre. Evidential hybrid re-sampling for multi-class imbalanced data. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 612–623. Springer, 2022.

[134] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from imbalanced data sets*, volume 10. Springer, 2018.

[135] Minlong Lin, Ke Tang, and Xin Yao. Dynamic sampling approach to training neural networks for multiclass imbalance classification. *IEEE Transactions on Neural Networks and Learning Systems*, 24(4):647–660, 2013.

[136] Xulei Yang, Qing Song, and Yue Wang. A weighted support vector machine for data classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(05):961–976, 2007.

[137] Hussein A Abbass, Charles S Newton, and Ruhul Sarker. *Heuristic and optimization for knowledge discovery*. IGI Global, 2001.

[138] Wei Feng. *Investigation of training data issues in ensemble classification based on margin concept: application to land cover mapping*. PhD thesis, Université Michel de Montaigne-Bordeaux III, 2017.

[139] Philipp Probst. *Hyperparameters, tuning and meta-learning for random forest and other machine learning algorithms*. PhD thesis, lmu, 2019.

[140] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[141] Markus Olsson, Simon Malm, and Kasper Witt. Evaluating the effects of hyperparameter optimization in vizdoom, 2022.

[142] T Ryan Hoens and Nitesh V Chawla. Imbalanced datasets: from sampling to classifiers. *Imbalanced learning: Foundations, algorithms, and applications*, pages 43–59, 2013.

[143] Kanae Takahashi, Kouji Yamamoto, Aya Kuchiba, and Tatsuki Koyama. Confidence interval for micro-averaged f1 and macro-averaged f1 scores. *Applied Intelligence*, 52(5):4961–4972, 2022.

[144] Juri Opitz and Sebastian Burst. Macro f1 and macro f1. *arXiv preprint arXiv:1911.03347*, 2019.