FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Tuomas Varanka

FACIAL MICRO-EXPRESSION RECOGNITION WITH NOISY LABELS

Master's Thesis
Degree Programme in Computer Science and Engineering
September 2020

Varanka T. (2020) Facial Micro-Expression Recognition with Noisy Labels. University of Oulu, Degree Programme in Computer Science and Engineering, 52 p.

ABSTRACT

Facial micro-expressions are quick, involuntary and low intensity facial movements. An interest in detecting and recognizing micro-expressions arises from the fact that they are able to show person's genuine hidden emotions. The small and rapid facial muscle movements are often too difficult for a human to not only spot the occurring micro-expression but also be able to recognize the emotion correctly. Recently, a focus on developing better micro-expression recognition methods has been on models and architectures. However, we take a step back and go to the root of task, the data.

We thoroughly analyze the input data and notice that some of the data is noisy and possibly mislabelled. The authors of the micro-expression datasets have also acknowledged the possible problems in data labelling. Despite this, no attempts have been made to design models that take into account the potential mislabelled data in micro-expression recognition, to our best knowledge. In this thesis, we explore new methods taking noisy labels into special account in an attempt to solve the problem. We propose a simple, yet efficient label refurbishing method and a data cleaning method for handling noisy labels. We show through both quantitative and qualitative analysis the effectiveness of the methods for detecting noisy samples. The data cleaning method achieves state-of-the-art results reaching an $F1\text{-}score$ of 0.77 in the MEGC2019 composite dataset. Further, we analyze and discuss the results in-depth and suggest future works based on our findings.

Keywords: Affective computing, Facial expressions, Noisy data, Machine learning

Varanka T. (2020) Kasvojen mikroilmeiden tunnistus kohinaisilla luokilla. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 52 s.

TIIVISTELMÄ

Kasvojen mikroilmeet ovat nopeita, tahattomia ja pienen intensiteetin omaavia kasvojen liikkeitä. Kiinnostus mikroilmeiden tunnistamisesta johtuu niiden kyvystä paljastaa henkilöiden todelliset piilotetut tunteet. Pienet ja nopeat kasvojen lihasten liikkeet eivät olet pelkästään vaikeita huomata, mutta oikean tunteen tunnistaminen on erittäin vaikeaa. Lähiaikoina mikroilmetunnistusjärjestelmien kehitys on painottunut malleihin ja arkkitehtuureihin. Me kuitenkin otamme askeleen taaksepäin tästä kehitystyylistä ja menemme ongelman juureen eli dataan.

Me tarkastamme käytettävän datan huolellisesti ja huomaamme, että osa datasta on kohinaista ja mahdollisesti väärin kategorisoitu. Mikroilmetietokantojen tekijät ovat myös myöntäneet mahdolliset ongelmat datan kategorisoinnissa. Tästä huolimatta meidän parhaan tietomme mukaan mikroilmeiden tunnistukseen ei ole kehitetty malleja, jotka huomioisivat mahdollisesti väärin kategorisoituja näytteitä. Tässä työssä tutkimme uusia malleja ottaen virheellisesti kategorisoidut näytteet erityisesti huomioon. Ehdotamme yksinkertaista, mutta tehokasta oikaisu menetelmää ja datan puhdistus menetelmää kohinaisia luokkia varten. Näytämme sekä kvantiviisisesti että kvalitatiivisesti menetelmien tehokkuuden kohinaisten näytteiden havaitsemisessa. Datan puhdistus menetelmä saavuttaa huippuluokan tuloksen, saaden $F1$-*arvon* 0.77 MEGC2019 tietokannassa. Lisäksi analysoimme ja pohdimme tuloksia syvällisesti ja ehdotamme tutkimuksia tulevaisuuteen tuloksistamme.

Avainsanat: Affektiivinen laskenta, Kasvojen ilmeet, Kohinainen data, Koneoppiminen

TABLE OF CONTENTS

FOREWORD

This work has been done at the Centre for Computer Vision and Signal Analysis (CMVS). I would like to thank Professor Guoying Zhao for the opportunity to work in her research group and for the great support and guidance she provided during the thesis. A huge thanks also goes to Wei Peng for helping in both the technical and written side of the thesis.

Oulu, September 11th, 2020

Tuomas Varanka

LIST OF ABBREVIATIONS AND SYMBOLS

AU	Action Unit
CASME	Chinese Academy of Sciences Micro-Expressions
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FPS	Frames Per Second
HS	High Speed
IDC	Initial data Cleaning
ILC	Iterative Label Correction
LBP	Local Binary Pattern
LTM	Loss Thresholding with Moments
LOSO	Leave-One-Subject-Out
MDMO	Main Directional Optical Flow
ME	Micro-Expression
MEGC	The Facial Micro-Expressions Grand Challange
MEVIEW	Micro-Expression VIdEos in the Wild
NAS	Neural Architecture search
NMER	Neural Micro-Expression recognizer
OF	Optical Flow
OFF-ApexNet	Optical Flow Features from Apex frame Network
RCN	Recurrent Convolutional Network
RGB	Red, Green, Blue
ROI	Region of Interest
SAMM	Spontaneous Actions and Micro-Movements
SELFIE	SELectively reFurbIsh unclEan samples
SMIC	Spontaneous Micro-Expression Corpus
SSSNet	Shallow Single Stream network
STSTNet	Shallow Triple Stream Three-dimensional Network
SVM	Support Vector Machine
TIM	Temporal Interpolation Model
TOP	Three Orthogonal Planes
UAR	Unweighted Average Recall
UMAP	Uniform Manifold Approximation and Projection

$\mathcal{C}$	The set of clean labels
f	A neural network
FN	False Negative
FP	False Positive
$F1$	$F1\text{-}score$
$\mathcal{L}$	Loss function
$\mathcal{R}$	The set of refurbished labels
$\mathbf{T}$	Transition matrix
TP	True Positive
V_x	Movement vector to direction x
V_y	Movement vector to direction y
$\mathbf{x}$	Data

y	Label
$\tilde{y}$	Noisy label
$\hat{y}$	Predicted label
α	Noise rate for loss thresholding with momentum
ϵ	The optical strain
$\boldsymbol{\theta}$	Parameters of a neural network
μ	The sample mean
σ	The sample standard deviation

1. INTRODUCTION

Humans are often the subjects of many scientific studies, and even the subjects of entire fields. In the field of artificial intelligence, more specifically machine learning and computer vision, we are also interested in humans. Countless methods have been developed to recognize different properties from humans, *e.g.*, face recognition, speech recognition, natural language understanding, recommendation systems and affective computing. Affective computing is the study of emotional behaviour of humans, including the emotional analysis of facial expressions and micro-expressions, speech and gestures—each a way of communicating non-verbally. A simple example of a facial expression is a smile, which often means that a person is feeling positive or happy about a situation. However, this is not always the case as is shown by Ekman and Friesen [1]. In 1969, the two psychologists, Ekman and Friesen, analyzed a filmed interview of a psychiatric patient. The patient seemed genuinely happy and not suicidal throughout the whole video, indicating her feelings further by using non-verbal communication in the form of a smile. However, when reviewing the tapes later Ekman found a brief segment of anguish on her face in the form of an micro-expression. It was later confirmed that she was indeed suicidal.

Micro-expressions (MEs) are rapid, involuntary and low intensity facial movements [2]. The duration of a typical ME lasts from 1/25 of a second to 1/2 of a second [2]. The intensity of MEs is very low and subtle (See Figure 2). The involuntary characteristic of an ME means that a person may perform an ME,

1. even if they are not trying to carry out any expressions
2. even if they are actively trying to hide giving away any expressions
3. even if they are unaware of expressing any emotions.

It has been shown that MEs are able to show the true emotions of a person [2], which has many applications and makes researching them particularly useful. An immediate application of MEs was shown by Ekman in the medical domain, further applications can be found from fields such as healthcare and well-being, business negotiations, security systems and marketing research [2].

The usefulness of being able to reliably recognize MEs are immense, but recognizing them has been found to be especially difficult [2]. For a person to be able to spot MEs accurately, they would have to be trained for the task. Typical humans lack the training and are thus unable to effectively detect MEs. In an attempt to make MEs more accessible, Ekman created a micro-expression training tool to train people to recognize MEs better [2]. However, the performance after training on the tool for humans was only around 50% accuracy and the training of a single person is costly. Since recognizing MEs for humans is such a difficult task, even after training, we hope to use an alternative method.

Automatic micro-expressions analysis systems use state-of-the-art techniques from machine learning and computer vision in order to reliably recognize MEs. Machine learning systems require carefully annotated data in order to perform reliably and with high performance. As MEs are difficult for even humans to spot, producing a dataset for the task is particularly difficult, as the dataset has to be labelled by humans.

Due to the characteristics of MEs (rapidness, involuntary and low intensity) not only is the data collection process difficult, but so is the process of labelling the video sequences. The characteristics of MEs make the labeling process especially volatile to mistakes. By analyzing samples from an already collected dataset we were able to find potentially mislabelled samples. In addition, the authors who created the datasets [3, 4] acknowledge the problematic labelling process and that it is prone to mistakes. Despite this no further investigations about the legitimacy of the labels or a model that is able to resolve the problem have been developed, although some papers do mention the problem [5, 6]. We hypothesize that there may be incorrectly labelled samples in the datasets, which are often referred to as noisy labels in the literature.

In recent years deep neural networks (DNNs) have achieved state of the art in many fields of computer vision, *e.g.*, face recognition and object detection. The superior performance of DNNs has also been demonstrated in ME recognition as state-of-the-art results are almost exclusively from DNNs. Deep neural networks provide immense classification performance, so far that they are even able to fit datasets with completely random labels [7]. The testing performance is however equivalent to random assignment as the network is not able to generalize over the training data. To combat the fitting to noise, many regularization techniques have been proposed to alleviate the problem, such as dropout, batch normalization, early stopping and weight decay [8]. However, the regularization methods are often targeted to noisy data in general and not specific to label noise, which can severely reduce the performance of the system [9].

Noisy label methods have recently gained interest in computer vision [10]. Many datasets have mislabelled samples, due to the massive size of the dataset or the difficulty and ambiguity of the samples [11]. As networks are very prone to overfitting and memorizing noisy samples, techniques specific to regularizing the noise contained in labels have been developed. These methods often manipulate the loss function by weighting it, by matching the distribution of noisy and clean labels, by discarding samples or by correcting labels [11]. To the best of our knowledge there have been no previous attempts at employing noisy label methods to micro-expression recognition. Many of the noisy label methods found in the literature are however often designed for large datasets with a high percentage of noisy labels. In addition the methods are often validated on synthetically generated noise, which has been shown to be different from real world noise [12]. We thus propose noisy label methods that are capable of dealing with the challenges of micro-expressions. We propose simple, yet effective methods to find mislabelled samples and ignore the incorrectly labelled samples during the training procedure, helping the network avoid training on noisy labels.

This thesis is structured as follows. Chapter 2 introduces related work to MEs. The chapter includes preliminaries of facial-expressions, ME datasets and ME recognition methods from the literature. Chapter 3 introduces the noisy label problem, methods that have been developed to solve the noisy label problem and finally we introduce our methods for noisy labels. In Chapter 4 we give details about the implementation of the methods, the used models and evaluation, including metrics and protocols. Chapter 5 showcases the results obtained and an in-depth analysis from the results is performed. Finally we conclude the thesis in Chapter 6 by a summary.

2. MICRO-EXPRESSIONS

In Chapter 1 we referred to the analysis of MEs interchangeably with the terms *detecting*, *recognition*, *spotting* and *analyzing*. However, we make a distinction between those words from now on. Both *spotting* and *detection* refer to the task of finding the segment in which the ME is occurring. The starting point of an ME is referred to as the *onset*, while the ending point of an ME is referred to as the *offset*. We also refer to *apex* as the point in which the intensity of an ME is at its peak. The term *recognition* on the other hand refers to the classification of the segmented ME clip to its corresponding class. An automatic ME system showcased in Figure 1, consists of two tasks: first it *spots* the segment where the ME is occurring and then it *recognizes* the emotion by classifying it to one of the emotion classes. These tasks are often considered separate from each other due to their complexity. This thesis will only focus on the recognition task of MEs.

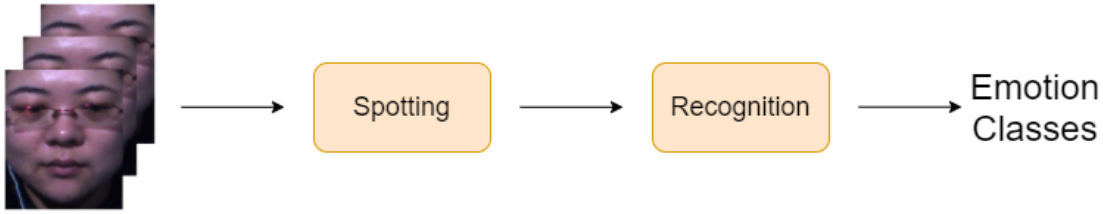


Figure 1. This figure showcases a general high level view of an ME analysis system. The system takes as input a sequence of videos (left). Spotting is performed to find the subset of frames where the ME is occurring. Recognition then utilizes the information from spotting to classify the emotions to their corresponding classes. The number of emotion classes and what emotion classes are used depend on the datasets. Both of the spotting and recognition steps are typically complex systems including preprocessing and many other steps. The example frames on the left are from the SMIC (defined later) dataset and the images are shown with the permission of the authors of [13].

2.1. Preliminaries for Facial-Expressions

Facial-expressions (FEs) are facial movements that convey emotions, they can be further categorized into macro-expressions and micro-expressions. Macro-expressions typically last between 0.5 and 4 seconds and they often have a large intensity, making the spotting of them easy for humans. In fact, macro-expressions are the facial-expressions everyone is familiar with and is in contact with every day. When compared to micro-expressions, there is a clear distinction between the two. MEs only last a maximum of 0.5 seconds [1], while macro-expressions go all the way to 4 seconds. The intensity of macro-expressions is large, while in MEs it is limited. Macro-expressions are mostly voluntary and they can be posed relatively easily compared to MEs, where it has been shown that posing is difficult [14]. These characteristics give us set of

[1]The duration of MEs is debatable, as some sources define the accepted length to be only 1/5th of a second [2].

Table 1. A summary of statistics from ME datasets

Dataset	Subset	Subjects	Samples	FPS	Resolution	Classes
SMIC [13]	HS	16	164	100	640×480	3
	VS	8	71	25	640×480	
	NIR	8	71	25	640×480	
CASME [17]	A	7	100	60	1280×720	8
	B	12	95	60	640×480	
CASME II [3]		26	247	200	640×480	5
CAS(ME)^2 [18]	B	22	57	30	640×480	4
SAMM [4]		32	159	200	2040×1088	7
MEVIEW [19]		16	31	25	1280×720	5

differences between the two, but in reality distinguishing between the two can be difficult.

When we think of emotions, we often think in terms of categories such as happy or sad. Formally, there are seven universal emotions: *happy*, *sad*, *anger*, *fear*, *surprise*, *disgust* and *contempt* [15]. How about a big smile and a small smile, how are these distinguished? Another common way to measure emotion is in terms of a continuous scale. In a dimensional emotion model, there are two axes, *arousal* (the level of affective activation) and *valence* (the level of pleasure) that measure the extent of these values from emotions [16]. For example, both happiness and anger have similar values in arousal, but have significantly different values in valence. Continuous measurement avoids the ambiguities involved with discretizing values. However, in MEs due to the complexity and difficulty in measuring intensities, the discrete categories are commonly used instead.

Action units (AUs) are based on FACS (Facial action coding system) [2] which classify different facial muscle movements on a person's face. For example, happiness is a combination of AU6 (cheek raiser) and AU12 (lip corner puller). AUs can be used to label images with less ambiguities as an emotion is composed of multiple smaller facial movements. However, it has been noted that in MEs this correspondence is not so straightforward [4]. Two distinct MEs with the same AUs may still be considered as different emotions due to self-reports and the content of the elicited material in the datasets.

2.2. Micro-Expression Datasets

As mentioned in Chapter 1, collecting ME data is difficult, and labeling the data correctly is an even harder challenge. Thus, a big emphasis is given to the datasets in this thesis. There are a total of six publicly available spontaneous ME datasets as of writing this thesis. Recently a composite dataset, MEGC2019 [20], was formed to provide a more generalizable way of evaluation. The MEGC2019 consists of a total of three datasets from the six publicly available ME datasets. A bigger emphasis is given to the three datasets as MEGC2019 will be used in the evaluation phase. A summary of all the datasets can be seen from Table 1 and Table 2.

First attempts at creating ME datasets consisted of posed MEs, where a group of actors tried to imitate MEs by attempting to do quick and low intensity facial-

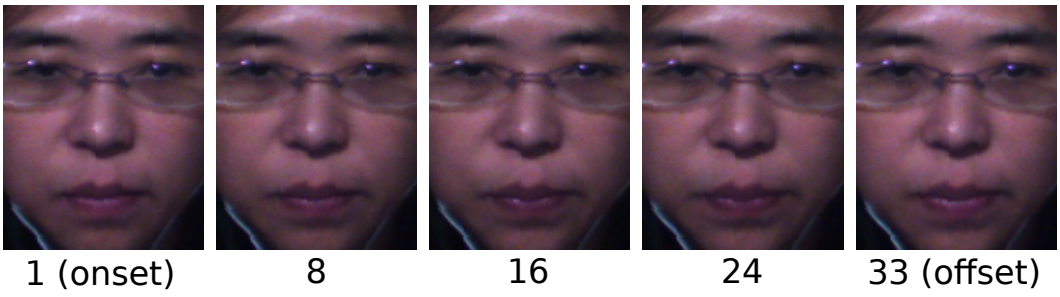


Figure 2. Example frames from the SMIC dataset from subject one showing *surprise* emotion. The numbers below correspond to the frame number. Images shown with the permission of the authors of [13].

expressions. However, the acted expressions were found to have for example different characteristics compared to spontaneous MEs [21]. Therefore, the focus has shifted towards collecting spontaneous MEs. Unfortunately, producing a spontaneous ME dataset is much harder than a dataset of posed MEs. The subjects need to be emotionally involved as acting is not enough. Emotional involvement of subjects is achieved by providing stimuli for the subjects, often through emotion inducing video clips. A high stake situation where the subjects are not allowed to show emotion is created to further increase ME inducement [13]. After collecting data comes the part we are especially interested in this thesis: labelling of the micro-expression emotion classes.

As mentioned earlier, the labelling process for MEs is difficult. First, the micro-expressions have to be detected, then the onset and offset frames have to be labelled, only after that can the emotion label be given. Different methods have been used for labelling the datasets. A convex combination of the following have mostly been used: labels based on AUs, self-report from the participant and the label from the inducing material. All the datasets used at least two of these techniques as only depending on one would be unreliable. Despite the use of a combination of different techniques, the labelling process still includes some ambiguities that could result in mislabels [4]. We will look at the labelling process of individual datasets in their corresponding sections. An important note to make is that AUs are mostly objective (label coders have some discrepancies) and the self-reports and the label from the inducing material are subjective. We hypothesize that this is one of the main reasons for ambiguities in the labels.

2.2.1. Spontaneous Micro-Expression Corpus (SMIC)

SMIC [13] was the first ME dataset that was not posed, but rather contained spontaneous MEs. It is an extension of the original SMIC-sub [22]. The SMIC-sub contains 77 samples from six different subjects recorded using a high speed camera at 100 FPS (frames per second). The extended version includes more samples as well as two more domains, a normal camera at the rate of 25 FPS and a near infrared camera

at 25 FPS as well. The extended version contains a total of 164 samples with all the samples recorded at a resolution of 640×480. It only uses three classes to distinguish the emotions: *positive*, *negative*, and *surprise*, as they found the labelling process to be ambiguous and labelling to six different classes difficult. An example of the dataset can be seen in Figure 2. The distribution of the samples can be seen in Table 2.

The dataset does not contain labelled AUs nor the location of the apex frame, although an extended version SMIC-E [14] was published with the locations of the apex frames. The labelling of emotions was done by two annotators that compared their results with the participants' self-reports. Only the samples in which both the annotators agreed with the participants' self-reports were included in the dataset. The stimuli selected was the same for all the participants and no individualized selections were made.

2.2.2. Chinese Academy of Sciences Micro-Expression II (CASME II)

CASME II [3] is a continuation for the CASME [17] dataset, the datasets however contain separate samples. CASME II uses a high frame rate of 200 FPS and does not contain flickering lights, as in the case of SMIC. A total of 247 samples were collected from 26 subjects. A spatial resolution of 640×480 was also used. As opposed to SMIC, CASME II labels the emotions to five different classes: *happiness*, *disgust*, *surprise*, *repression*, and *others*. The main reason for this is that CASME had a total of eight classes with some of the classes only having a few samples, so they were all put into the *others* class, to avoid an imbalanced setting.

The labelling was performed by two annotators who first marked the AUs and then determined the corresponding emotions from them. The final label is given by a combination of the AUs, participants' self-reports and the content of the inducing material. How the combination is done exactly is not however discussed. The authors acknowledged that the labelling is ambiguous and debatable. For example a clip of *chewing worms* was found to be disgusting, funny or interesting for different people. Further, the two annotators had an reliability value of 84.6% when annotating the AUs. This is also a potential cause of the unreliable labelling.

2.2.3. Spontaneous Actions and Micro-Movements (SAMM)

SAMM [4] is the newest ME dataset contained in the MEGC2019, published in 2016. It has a high temporal dimension of 200 FPS, as well as a large spatial dimension of 2040×1088. SAMM improves on the distribution of its participants by having people from 13 different ethnicities and a wider distribution in the participants' age. Having wider distributions weakens the possibilities of biases and can help create a more generalizable model. The samples are labelled based on the seven basic emotions. To prevent a problem that occurred in CASME II with some clips inducing different emotions for different people, the videos were selected personally. The participants answered a questionnaire before the experiment so that the video clips could be chosen to be more effective.

Table 2. Distributions of emotion classes from ME datasets

SMIC [13]			
Subset	Positive	Negative	Surprise
HS	51	70	43
VIS	28	23	20
NIR	28	23	20

CASME [17] A & B							
Contempt	Disgust	Fear	Happiness	Repression	Tense	Sadness	Surprise
3	88	2	5	40	28	6	20

CASME II [3]				
Happiness	Disgust	Repression	Surprise	Others
33	60	27	25	102

CAS(ME)^2 (Subset B) [18]			
Positive	Negative	Surprise	Other
8	21	9	19

SAMM [4]							
Anger	Contempt	Fear	Disgust	Happiness	Other	Sadness	Surprise
57	12	8	9	26	26	6	15

MEVIEW [19]				
Anger	Contempt	Fear	Happiness	Surprise
1	6	6	8	9

The labelling is performed by three annotators instead of the typical two to ensure a more reliable result. The inter-coder reliability is 82% for AUs, slightly less than in CASME II. Unlike in SMIC and CASME II, the self-reports conducted in SAMM were done beforehand. The participants' fears and joys were asked beforehand in order to provide them a suitable video, which was then used to label the samples. The authors also state that the focus of the dataset is on the annotated AUs and not on the emotion classes as their reliability is questionable, therefore calling the dataset not a micro-expression dataset but rather a *micro-facial movement* dataset.

2.2.4. MEGC2019

Due to the low number of samples in each of the datasets, the imbalanced distributions of emotions and the inconsistent evaluation protocols used by the community, a composite dataset was suggested in the second facial micro-expression grand challenge (MEGC2019) [20]. The combined dataset also evaluates the models ability to generalize over multiple different domains. Since the proposal, the MEGC2019 has been well received and is used commonly for a fair evaluation of proposed methods.

MEGC2019 consists of three datasets, SMIC, CASME II and SAMM. As all of the datasets have introduced different classes for emotions, a united set of classes had to be proposed. As SMIC has the least classes, the three classes from it were selected. The class labels from CASME II and SAMM that were labelled as *repressions*, *anger*, *contempt*, *disgust*, *fear* or *sadness* were changed to *negative*. The samples labelled as *happiness* were changed to *positive*. The *others* class from CASME II and SAMM

Table 3. Distributions of emotion classes for MEGC2019

Emotion class	Positive	Negative	Surprise	Total
SMIC	51	70	43	164
CASME II	32	88	25	145
SAMM	26	92	15	133
Combined	109	250	83	442

were omitted. SMIC kept all of its 164 samples; while CASME II lost over 102 samples, only keeping 145 in total; SAMM lost 26, totalling to 133 samples altogether. Thus, the new dataset consists of 442 samples from 68 different subjects with three different classes. The distribution of emotions can be seen from Table 3.

MEGC2019 is able to reduce the problem of small sample size, but even the 442 samples is still quite a low number when compared to the typical datasets used in deep learning such as CIFAR-10 [23] with 60000 samples and ImageNet [24] with more than 14 million samples. The class imbalance is also not as severe with the MEGC2019 compared to the single datasets. However, combining the datasets creates another problem: domain shift. Domain shift is the problem where the distribution of testing data differs significantly from the training data. Now that the composite dataset consists of three smaller datasets, the model may learn domain specific information which may lead to lower generalization ability.

2.2.5. Other Datasets

Other ME datasets not used in our study include: CASME [17], CAS(ME)^2 [18] and MEVIEW [19]. CASME is the predecessor of CASME II which includes a total of 195 samples with a varying resolution depending on the subset. The frame rate is at 60 FPS and it includes a total of eight classes. A major reason why CASME is less frequently used is the imbalanced classes, which was later fixed in CASME II. The CAS(ME)^2 is not used as commonly as it contains both macro- and micro-expressions, and the ME part only includes a total of 57 samples captured at only 30 FPS. MEVIEW is the newest dataset published in 2017 and it differs from the other datasets as the samples are not from laboratory conditions, but rather collected from online videos. As the clips are from real life situations the clips contain challenging variables, such as a changing angle from the camera and the changing lighting, providing a realistic scenario of real life challenges. The dataset also has a very small size of only 31 samples.

2.3. Micro-Expression Recognition Methods

Recognition methods in MEs are comprised of two categories as in many other tasks, from traditional methods and deep learning methods. By traditional methods we refer to methods that consist of feature extraction by a hand designed method and a classifier, whereas deep learning methods learn the feature extraction by themselves and usually the feature extraction and classification is combined. Due to the limited amount of samples in ME datasets, traditional methods have been preferred as they often require much less samples for learning, however with the development of new deep learning

techniques NNs are often the preferred choice. The main two methods for traditional methods in ME recognition can be further categorized to appearance based methods and motion based methods.

2.3.1. Appearance Based Methods

Appearance based methods use the intensity of pixel values to create features that are then used for classification. Gabor filters are a collection of different filters with different orientations and shapes. They were used by [25] for ME recognition and spotting. In [14] a histogram of oriented gradients (HGO) was used. The HOG first calculates the gradients of the image for blocks, which are then transformed to the polar coordinates. A histogram is then created by quantizing the direction, and the value for each bin is then created by the sum of magnitudes of gradients.

Local binary pattern (LBP) is a texture descriptor that calculates a feature vector by thresholding the neighboring pixels in a circular area from divided blocks from an image. Since MEs occur both in the spatial and temporal domains, a dynamic feature extraction method is required. To capture the temporal domain as well as the spatial domain an extended version of the LBP is used, local binary pattern from three orthogonal domains (LBP-TOP) [26]. As the name suggests LBP-TOP calculates the LBP from three different planes, the planes being XY, XT and YT. Here X and Y refer to the x-coordinate and y-coordinate in an image, and T to the time dimension, *i.e.*, the video length. Only calculating the XY planes would result in the original LBP.

LBP-TOP was used as the baseline in the first spontaneous ME dataset, and has since become a stable baseline for comparison as well as inspired many variants [27, 28, 29]. In [13] a temporal interpolation model was used to both downsample and upsample the samples to have an equal number of frames, after which the LBP-TOP was used for feature extraction, finally the features were then classified using the support vector machine. LBP-SIP (local binary pattern - six intersection points) [27] only uses the six intersection points unique in the different planes to reduce the computational cost. LBP-MOP (local binary pattern - mean orthogonal planes) [28] uses the mean of the three different planes instead of the individual frames to reduce computations and redundant information. Extended LBPTOP (ELBPTOP) [29] uses the second order information from radial and angular differences to include extra information.

2.3.2. Motion Based Methods

The second type of commonly used traditional method in ME recognition is based on the optical flow. We first define the optical flow (OF) precisely as it is used extensively throughout our analysis. OF is used to measure the difference between two sequential images. The brightness constancy constraint

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{1}$$

tells us that the intensity of an image $I(x, y, t)$ stays constant throughout different time steps. The intensity change of $I(x, y, t)$ at pixel (x, y) at time t can be thought as adding the movement Δx and Δy in duration Δt. By assuming the movement is small, equation (1) can be constructed with the Taylor series by linearizing the right side

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0, \tag{2}$$

where V_x is the movement in x direction and similarly V_y is the movement in y direction [21]. Equation (2) is problematic since we have two unknown variables and only one equation. Different methods for approximating optical flow have been developed. A simple method is to assume that the neighboring points share the same changes of V_x and V_y—this changes the problem to an overdetermined linear system which can be solved easily by least squares [30]. More advanced techniques that use less assumptions and perform better have been developed. In this thesis the OF method used is described in [31].

In addition to the $[V_x, V_y]$ components we can calculate the optical strain ϵ for second order information [32]. The optical strain is defined as

$$\epsilon = \frac{1}{2}[\nabla V + (\nabla V)^T], \tag{3}$$

where $V = [V_x, V_y]$. The optical strain is the gradient of the OF components, which is able to give more detailed information by disregarding noisy parts.

Methods

The first use of OF in ME recognition was by histogram of oriented optical flow (HOOF) [33, 21]. HOOF calculates the movement vectors in the Cartesian coordinates $[V_x, V_y]$ and transforms them to the polar coordinates $[\rho, \theta]$. The orientation directions are then placed to bins and the magnitudes corresponding to the orientations from each of the bins are summed together. The bins can then be used as feature vectors. Main directional mean optical flow (MDMO) [21] improves HOOF by introducing regions of interest (ROIs) from which the histograms are calculated. The 36 ROIs were carefully constructed to include most of the AUs in such a way that one ROI would correspond to one AU. MDMO reduces the number of bins by only choosing the bin with the maximum number of vectors and then calculates the mean vector from that. SparseMDMO [34] avoids losing structure by aggregating over the frames, and instead uses all histograms from all frames. This however increases the length of the feature vector, which is then made sparse by using graph sparse coding [35] to reduce the feature vector to only include the important features. Bi-weighted oriented optical flow (Bi-WOOF) [36] takes a slightly different approach by only using the onset and the apex frame of the sample. In addition to using the $[V_x, V_y]$ components, Bi-WOOF also calculates the optical strain to weigh the ROIs (different from MDMO).

2.3.3. Deep Learning Methods

Many recent methods in ME recognition have started to adapt deep learning into their methods [37, 15, 38, 39, 32, 40, 41, 42]. As with most computer vision tasks, ME recognition also uses convolutional neural networks (CNNs). Due to the temporal component of MEs, a sequential model is required to model videos, such as long short-term memory networks (LSTM). A combination of LSTM and CNN was developed in [15]. However these methods require a large number of parameters, which is something that should be avoided with small scaled datasets. Instead, the finding from [36] of only using the onset and apex frame was adopted—thus only requiring the use of a typical CNN. One of the first uses of CNNs on only the apex was in [38], where the authors fine-tuned a VGG-face on the apex, where motion of the experessions had been magnified by a Eulerian magnification method. In addition, they used the neighboring frames of the apex in the training phase to include more data.

One of the first approaches of using a CNN on OF inputs was the work from optical flow features from apex frame network (OFF-ApexNet) [39]. The OF is able to capture the major movements, reducing the need for multiple frames and even the need for high resolution inputs. In OFF-ApexNet, a network with two convolutional streams is provided for each of the input components $[V_x, V_y]$ and then the learned features are concatenated to fully connected layers for prediction. The inputs are reshaped to a size of only 28×28, to reduce the computational complexity and the number of required parameters. The network consists of two convolutional layers with max pooling at the end of each stream and three fully connected layers for classification. Shallow triple stream three-dimensional CNN (STSTNet) [32] has a similar architecture to OFF-ApexNet, but only uses a single convolutional layer and a single fully connected layer, which drastically reduces the number of parameters. The input is changed to include the optical strain $[V_x, V_y, \epsilon]$, increasing the number of streams to three.

An RCN (recurrent convolutional network) was employed in [40], as the result of finding that shallower lower complexity models perform better in ME recognition. In addition, it was found that the input size should also be relatively small (60×60), allowing to further decrease the number of parameters. The authors also developed three independent hyperparameter-free modules for the RCN: wide expansion, shortcut connection and attention unit. To find the best combination and location of the proposed modules, a neural architecture search was performed but the results did not improve over just having the individual modules.

To compensate for the lack of data, NMER (neural micro-expression recognizer) [41] proposes to increase the dataset by including data from a macro-expression dataset. Due to the difference between macro- and micro-expressions, two domain adaptation techniques are applied for the model to take full advantage of the extra data. As input data for the MEs, the raw frame from the apex is used. For the macro-expressions the middle frame between the onset and the apex is used to simulate the lower intensity of MEs. To maximize the similarities between the two domains a motion magnification method is used on the MEs. Adversarial domain adaptation is used in the training process to encourage domain-independent features. However, the model only performs at the level of LBP-TOP, possibly due to the innate differences between macro- and micro-expressions.

To compensate for the lack of data, [42] uses a GAN (generative adversarial network) to create more data. Due to the complexity of the raw data, only the optical flow is used. They experimented with two types of GANs: auxiliary classifier GAN and self-attention GAN. However, the performance of the models only decreased with the generated data. We hypothesize that since the GANs are used independently of the classifier they are optimizing for the images ability to look natural. However, this is not optimal as shown in [43], instead the GAN should be optimized in an end-to-end manner with the classifier such that the generated images are optimized for the classification task at hand.

3. NOISY LABELS

As machine learning is being presented with more and more tasks, it is also being introduced to more difficult tasks. Some of the tasks can be so difficult that even a human is not able to consistently perform well on them. This can make the labelling process of a dataset a demanding task, which is a crucial step as typical machine learning systems require well annotated large scale data. The difficulty of the tasks may be due to the innate ambiguity of the samples, the samples being borderline cases or the samples having multiple correct labels. Examples of tasks that contain difficult samples are many medical imaging problems such as segmentation of brain lesion [11] and of course micro-expressions. As annotators are labelling the data, the ambiguous data may lead to mislabelled samples, which we refer to as noisy label samples. These are different from samples where the noise is contained in the input data. A simple example of a noisy label is an image of a cat that has been labelled as a dog.

The recent success of deep learning has led to its widespread usage over many different tasks in machine learning, computer vision and even outside of these fields. One of the reasons is their incredible representation power. It was even shown that deep learning models are able to fit datasets with completely random labels [7]. Due to the powerful representation ability of DNNs, the incorrect labels can be detrimental to the training process compared to traditional machine learning methods [9]. To prevent DNNs from overfitting to the training set, various regularization methods are used: dropout, batch normalization, weight decay and early stopping [8]. However, these methods are meant for overall regularization of noisy data, and are not specifically designed to solve the problem of noisy labels. Thus an interest in developing noisy label methods that are able to regularize incorrect labels have surged recently [10]. The basic idea of noisy label methods is to regularize the effects of incorrectly labelled samples during the training process.

We divide the noisy label tasks in the literature to two different categories:

1. **Large scale datasets created by non-experts or automated systems.** One of the thriving forces of the success of deep learning can be attributed to the existence of large datasets. Creating a large dataset is desirable as it can increase the performance of models significantly [11]. However, the datasets have to be carefully annotated. To achieve this, multiple annotators are often used to avoid mistakes [11]. Using multiple annotators on a large dataset can however be expensive and thus cheaper alternatives have been sought [11, 44]. One solution is to use non-expert humans to annotate the dataset. Another, even cheaper solution is to use automated methods with little or no human supervision at all. An example of this is Webdata [11], which is created by using an internet crawler that annotates the data based on information around the data. Although these methods are cheaper alternatives than using multiple annotators, the labelling methods often suffer from high label noise.

2. **Difficult and ambiguous data.** This type of data is inherently difficult or ambiguous even for humans. The datasets are typically relatively small allowing it to be annotated by multiple people, but they can still contain noisy labels due to the difficulty of the data. In the previous section the datasets typically consist of images that are natural to humans, like animals or vehicles, but in this category

the data is not only difficult for non-experts but even for experts. The data can be ambiguous by being noisy, being a borderline case in between two classes or the sample may have multiple different labels that are all correct. Example datasets are medical images that require healthcare professionals trained for the task, and micro-expression data that also needs professional FACS coders and is not entirely objective. Due to the smaller size and the use of experts these datasets often have fewer mistakes compared to the automatically annotated datasets.

Based on [11] and [44] the two tasks seem to have slightly different methods, although the difference does not seem to be discussed frequently throughout the literature. Large datasets obviously have the advantage of having numerous training samples, which allows the use of larger models, validation data and the use of clean data (due to the simplicity of the data). Due to having numerous samples from which to learn, the discarding technique shown in Figure 5 can be used without necessarily dropping important samples that are crucial for training. In smaller datasets, discarding even a few samples can significantly affect the training negatively. Some datasets may also contain higher levels of noisy labels and thus require methods that are able to perform even with extremely high label noise.

3.1. Noisy Label Methods

We formally defined the noisy label problem here. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the space of inputs and the space $\mathcal{Y} = \{1, \ldots, c\}$ be the label space, where d is the dimension of the inputs and c is the number of classes. In a typical case we hope for the training set to consist of clean tuples of $(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y})$, but in a noisy label case we have $(\mathbf{x}, \tilde{y}) \in (\mathcal{X}, \tilde{\mathcal{Y}})$, where $\tilde{\mathcal{Y}}$ is the set of noisy labels. The clean and noisy labels are related by a function $g : \mathcal{Y} \rightarrow \tilde{\mathcal{Y}}$, that maps the clean samples to noisy samples. The task of noisy label methods is to try to undo the mapping, *i.e.*, find g^{-1}. However this is an ill-posed problem and only approximate solutions can be found. One set of methods tries to model g^{-1} on a class level by creating an error transition matrix $T_{ij} = p(\tilde{y} = j \mid y = i)$. Other methods settle for implicit solutions by discarding samples with noisy labels or refurbishing them through predictions.

Various methods have been proposed for learning with noisy labels, especially recently [10]. A small proportion of the methods were developed for traditional machine learning methods, which we will not cover. Due to NNs ability to fit noisy datasets, the problem of noisy labels is much more prevalent in deep learning methods than in traditional machine learning. We categorize the following subsections based on classification of the methods found in the literature inspired by [44, 11]. We deviate slightly from the categorization in the literature to better suit and emphasize the most important categories for ME recognition.

Before going into the methods we describe a crucial finding that is commonly used throughout the different methods, named as the *small-loss* trick by [44]. The loss of a single sample is determined by the DNNs prediction. For example, the frequently used categorical cross-entropy $-\sum_{n \in \mathcal{N}} y_n \log(\hat{y}_n)$ measures the difference between two distributions, where $\mathcal{N}$ is the set of possible classes. If the distribution of the predicted label $\hat{y}$ is similar to that of the real label y, the loss will be small. If the distributions are

different, then the loss will be high. For representative and unambiguous clean samples we would expect the loss to be small at the end of the training. For difficult clean samples the loss may be high as the network is not able to fully match the distributions. For samples with noisy labels we expect the loss to be high, as the network is predicting a distribution that does match the real label's distribution.

3.1.1. Loss Functions

The methods in this category are relatively self-explanatory, they simply change the loss function used to optimize the network. Mean absolute value of error (MAE) [45] uses the ℓ_1 norm to calculate the loss as opposed to the typical cross entropy. The idea is similar to performing regression using ℓ_2 and ℓ_1 norms, where the ℓ_2 weighs outliers heavily, while ℓ_1 keeps the weights equal for all samples. The use of MAE makes intuitive sense as samples with high loss values can be thought of as outliers or as samples with noisy labels in our case by the small-loss trick. However, the MAE loss can also discard clean samples that have a high loss, due to the samples merely being difficult, hindering the network's performance. A modified version that contains both the good qualities of MAE and cross-entropy was proposed to create a generalized cross-entropy by using a negative Box-Cox transformation [46]

$$\mathcal{L}_k\left(f(\boldsymbol{x}), \boldsymbol{y}_j\right) = \frac{\left(1 - f_j(\boldsymbol{x})^k\right)}{k}.$$

The authors in [46] show that the loss is equivalent to cross-entropy when k approaches 0 and MAE when $k = 1$. Then we can choose a k, such that $0 < k \leq 1$, to have a combination of the cross-entropy and MAE losses in order to have both the benefits of MAE and cross-entropy.

3.1.2. Label Cleaning

Label cleaning methods attempt to identify samples with incorrect labels before or during the training and either discard them or ignore their gradients during the training. One of the more intuitive ideas is to use the loss of a sample as an indicator based on the small-loss trick. Samples with high loss values are either discarded or ignored during the backpropagation and their gradients are not updated, while samples with small loss values are kept. One of the downsides of this method is that one has to define what is considered as a high loss value, which is dependent on the data. Frequently the notion of q-percentile is used, here q percent of the data is assumed to be noisy, and only the $(1 - q)\%$ is kept. The losses of a batch are sorted and $q\%$ of the highest values are discarded from updating their gradients. One of the downsides of q-percentile is the need for a warm-up, as the small-loss trick does not apply during the early stages of training. Nonetheless, the method is used by multiple works [11, 47, 48, 49] as either the primary method or as a part of a more complicated method. Another very intuitive idea is to use the network's predictions as a thresholding mechanism. Rank pruning [50] uses the confidence of the network

predictions (after the softmax layer) as an indicator whether the sample is noisy or not. This method makes the assumption of well behaving classification borders or "well calibrated" network [11]. A well calibrated network's predictions should match the likelihood of it being correct. However, this is often not the case as shown in [51, 52]. Networks frequently tend to be overconfident of their predictions, which makes this method reliant on the networks calibration level. CleanNet [53] uses a strategy that compares the feature representations of samples with noisy labels to samples with clean labels and then uses the similarity between the samples to determine whether the sample is kept or discarded. This method requires a clean subset of the data that can be used to create the clean feature representations.

An important thing to note is that all of the methods in this category can also be used to reweight the samples. Each of the samples can be given a weight instead of thresholding, samples with noisy labels should be given a low weight. The weights can be used to construct a sampling distribution which is used for sampling the data during training. Alternatively, the weights can be used to weigh the loss values, affecting the length and direction of each gradient step.

3.1.3. Label Refurbishment

Label refurbishment methods assign new pseudo labels to the samples, instead of completely discarding them. Correcting a label can however be risky—one may end up with more noisy labels than they started with if the corrected labels are wrong. The corrected pseudo label is attained by a convex combination $\lambda\tilde{y} + (1 - \lambda)\hat{y}$ of the noisy label $\tilde{y}$ and a predicted label $\hat{y}$. In [54] the authors use a static coefficient for the convex combination and find it through cross-validation. Different methods have been proposed to determine the coefficient of the convex combination. Note that the coefficient can be unique for each sample and it can be adaptive throughout the training. Dynamic bootstrapping [55] uses this idea and estimates the coefficient for each sample for every epoch using a beta mixture model, which measures whether the sample belongs to the distribution or not. The method is built around the idea that loss distribution often takes the shape of a beta distribution with clean data. SELFIE (SELectively reFurbIsh unclEan samples) [48] uses the simple idea that not all samples have to be refurbished. SELFIE first uses the basic q-percentile strategy to find samples with high loss values and discards them unless they satisfy the refurbishable condition. The refurbishable condition is met if the sample's predictions from the network are consistant. Formally the condition is given by the normalized entropy of the predicted values and a thresholding value. If the label satisfies the condition, its label is corrected by using the most frequent prediction from the network. SELFIE uses a $\lambda = 0$ and only uses the prediction $\hat{y}$ for the pseudo label.

3.1.4. Transition Matrices

Transition matrix methods modify the loss function or the predicted labels by multiplying it with the inverse transition matrix $\mathbf{T}^{-1}$ (defined in Section 3.1) in an attempt to make the distributions of the network's predicted probabilities match the

distribution of noisy labels. The idea was first proposed by [56] and they proposed to learn the transition matrix by adding the transition matrix as a linear noise layer to the end of the network and then backpropagating through it. To make the model converge, the authors use a trace norm of the transition matrix for regularization. Clean data can also be used to improve the estimation of the transition matrix. Backward correction [57] decouples the estimation of the transition matrix from the model and estimates it using either clean data or by using a pre-trained network. The pre-trained network gives the probabilities $p(\tilde{y} \mid \mathbf{x})$ for each class, which can be then used to construct the estimated transition matrix $\hat{\mathbf{T}}$.

Often datasets are labelled by multiple annotators and their inter-observer variability tells how well the annotators agreed with each other. A work by [58] shows that utilizing all the different sets of labels from different annotators leads to better performances compared to just using an aggregated version of the collection of labels. The authors estimate a confusion matrix for each annotator to model the true label distribution. Each of the confusion matrices are used with the noisy prediction to create a clean prediction for each annotator. The loss is then calculated as the sum of all the different predictions. Similarly to [56] a trace norm is used to regularize the transition matrices. In addition [11] experimented with the method on a task of prostate cancer digital pathology classification achieving the best result compared to just using a single annotator or a majority vote between the annotators.

More recently, T-revision [59] shows that clean samples are not necessary for the optimal learning of the transition matrix $\mathbf{T}$. They use the same idea as [57] of using the posterior probabilities $p(\tilde{y} \mid \mathbf{x})$ to estimate $\mathbf{T}$. However by only utilizing the noisy class posterior probabilities the transition matrix is often estimated poorly. As clean samples can significantly estimate $\mathbf{T}$ better, the authors propose to use samples that are likely to be clean. They determine the clean samples to be the ones with a high predictive value from the network. After the initial estimations of $\mathbf{T}$, revision are made by adding a slack variable $\Delta\mathbf{T}$ to the initial estimation $\hat{\mathbf{T}}$. Their proposed risk function is asymptotically equal to the risk function with clean data, ensuring accurate estimation.

3.1.5. Loss Reweighting

Loss reweighting attempts to give larger weights to clean samples and smaller weights to noisy labels in the loss function. Unlike methods like focal loss, AdaBoost and hard negative mining that give more weight to hard samples, the opposite is wanted in noisy label learning based on the small-loss trick [60]. Self-paced learning uses a decreasing weight function for the loss, giving more weight to samples with lower loss. However, this method requires a hyperparameter and the piecewise linear weighting functions may not be optimal for the used data. Thus, a multilayer perceptron is used to model the weighting function [60]. The weighting function is learned through a clean meta dataset. The authors show its ability to create complex weighting functions for different datasets. As the method models the weighting function, it can be used for both noisy label problems, as well as imbalanced data that requires the weights to be higher for high loss samples. The work from [47] uses an outlier detection method to weight the samples. Probabilistic local outlier factor (PLOF) gives a value between 0

and 1 whether the sample is an outlier or not and the authors use this value directly to weigh the samples. Conversely to thresholding, these methods can also be used for data cleaning, by setting a threshold value from the weights.

3.1.6. Training Procedures

Curriculum learning is a training strategy where a curriculum is created for the incoming data. The idea is natural to human learning—we start by learning easier things and then move to progressively harder tasks. Currently, most networks simply use a random order and the network might start learning difficult samples immediately. Although the idea of curriculum learning is often synonymous to starting with easier samples and moving to harder samples later on in the training, the idea can be generalized to creating an arbitrary curriculum that may in fact do the opposite. Another common strategy is to first train on "important" or representative samples, which is similar to the idea of easy samples. In [61] the authors design methods that are able to find representative samples, these samples are then used to create a ranking of the most representative samples which is then used as the curriculum.

Similar ideas have been used in noisy label methods as well. The idea is to rank the confidence level of each sample being a clean label and use the ranks to create a curriculum, the clean samples should be used first in the training and the incorrect labels later. MentorNet [62] uses a teacher network to guide the training of a student network by providing it a curriculum that is learned with an LSTM network. The curriculum is learned by using a small clean dataset. CurriculumNet [63] uses a clustering technique to find samples that are likely to have noisy labels. They cluster the data in a feature space provided by a pre-trained network and then sort the samples based on their local density to obtain the curriculum. This method is especially designed for massive web data, with heavy noise in addition to imbalanced data.

In sample selection the target is to select a subset of the minibatch that only contains the clean samples. Decouple [64] uses two networks and only updates the samples that have different predictions between the two networks. For easy clean samples the predictions of the two networks should match as both the networks will likely be able to predict the correct label. For noisy label samples the networks should also agree, the prediction is likely to be wrong, but both the networks agree. For difficult, near the decision boundary samples, the predictions could differ from the networks. With this strategy the networks are able to ignore the noisy labels and train using the difficult samples at the later stages of training.

Mixup [51] is not directly a noisy label method, but an overall regularization method. However, it has been observed to work as a regularizer for noisy labels as well [11]. Mixup works by synthesizing new labels using a convex combination of samples. Both the input data and the labels are combined as $\mathbf{X}_{mix} = \lambda \mathbf{X}_i + (1 - \lambda)\mathbf{X}_j$ and $y_{mix} = \lambda y_i + (1 - \lambda) y_j$, giving a soft label for the new sample. An extension of the work named Manifold Mixup [52] expands the use of convex combinations to include hidden layers as well. Both the methods have also been shown to improve results for domain generalization and zero-shot learning [65], demonstrating their capability as effective regularizers.

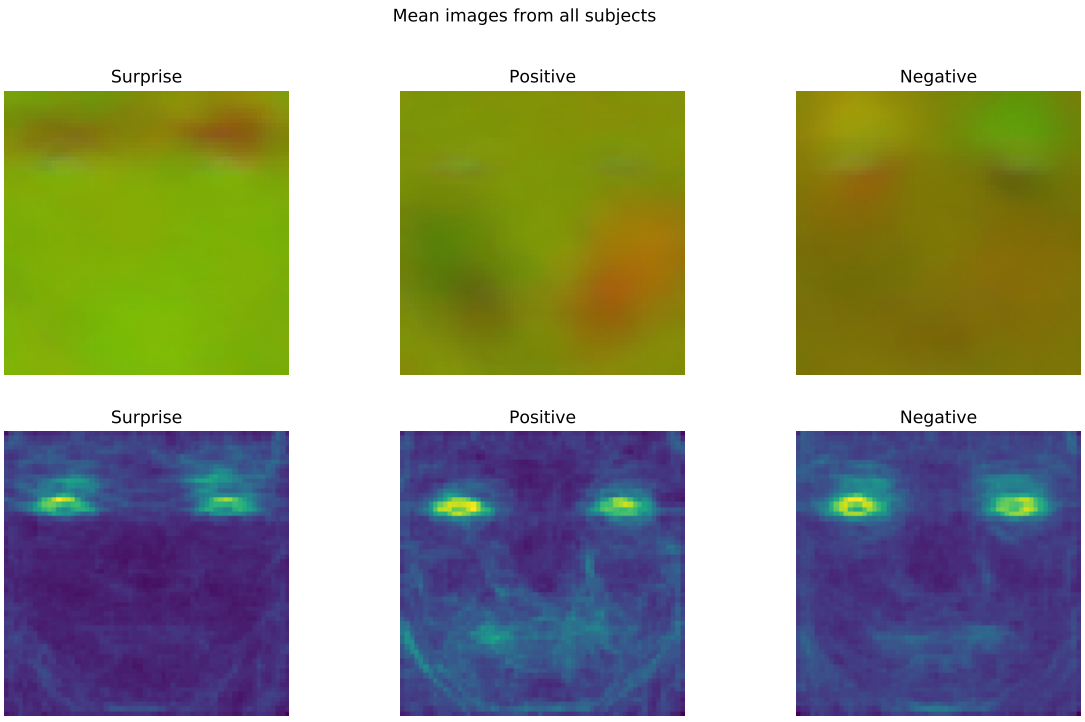


Figure 3. Mean of the optical flow aggregated based on the emotion class. Top: The three channels from OF $[V_x, V_y, \epsilon]$. The movement is shown in red. Bottom: Only the optical strain channel ϵ is displayed for clarity. The movement in shown in green and yellow.

3.2. A Look at the Micro-Expression Data

This section will provide a qualitative analysis of the ME data, where we will look at the the OF (optical flow) between the onset and the apex frame from the samples in the MEGC2019 dataset. Each sample consists of three channels: the horizontal component of OF, the vertical components of OF and the optical strain denoted by $[V_x, V_y, \epsilon]$.

Figure 3 shows the mean samples of each emotion class. The *surprise* emotion has mainl movement on the forehead and eyebrows. The *Positive* emotion mainly has movement on the left (from the participant) cheek. By looking at the optical strain the movement seems to be on both cheeks and the mouth area. The *Negative* emotion does not seem to have any distinct movements, there are some movements around the eyebrows, but also near the mouth. We believe this is due to the aggregated classes, as the MEGC2019 *negative* contains multiple other subclasses. The eyes are highlighted on all the classes due to blinking. We will use these distinct movements of each class to search for any inconsistencies in the dataset.

Figure 4 shows the first 15 samples of MEGC2019. Starting our observations with the sample zero located at the top left of the figure, we can immediately see that it is very noisy. By carefully inspecting it we are able to see some movement on the left cheek, which seems to correspond to the movements we saw in Figure 3 for the *positive* class, making the label for this sample likely to be correct. The next two samples, numbers one and two, are very similar and both seem to be quite noisy. Movements can

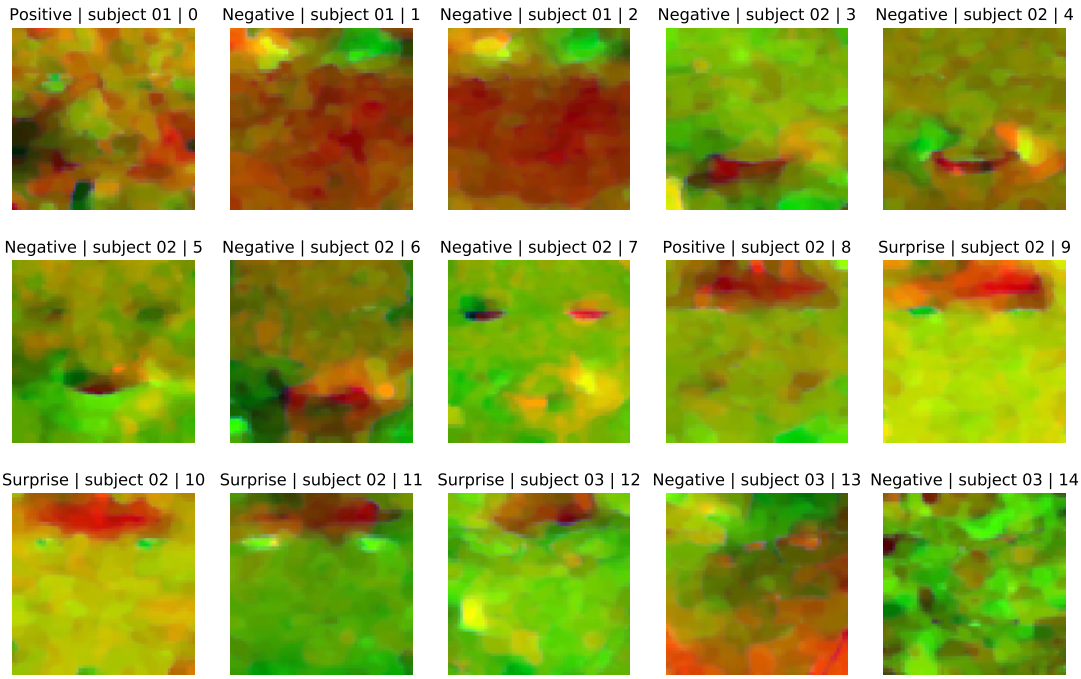


Figure 4. First 15 samples of MEGC2019. The title of each image consists of *the emotion class* | *the subject number* | *the sample number*.

be seen essentially everywhere below the eyes. These samples do not fit our description of *positive* as the movement is further spread from the cheeks or *surprise* as there is only small movement on the right eyebrow, therefore the class *negative* makes sense. Next, the samples three, four and five also contain many similarities with each other. These emotions are all *negative*, but they do not look anything like the samples one and two. This is due to the samples having a different subclass as the *negative* emotions were aggregated to one in MEGC2019.

An interesting discovery can be made from the samples eight, nine, ten and eleven. All of these samples look similar but one of them is labelled as *positive* while the others are labelled as *surprise*. The sample number eight also contains some movements on their cheeks which may have been the reason for the label *positive*, but so does sample number ten and it has not been labelled as *positive*. This is a great example of an ambiguous sample, where there could even be two correct labels, as the sample contains two distinct movements for two different emotions. However, confirming that this sample has a noisy label is difficult as we do not have an expert to confirm the finding and since this is only the apex of the OF, many more things could be happening in the other frames in the original domain. Nevertheless for the model performing the classification this sample will most likely be classified as *surprise* due to the distinct movements in the forehead. We find more similar samples by going through the dataset and hypothesize that these are samples that have been labelled incorrectly.

3.3. Proposed Methods

This section introduces three methods for noisy label learning: initial data cleaning (IDC), loss thresholding with moments (LTM) and iterative label correction (ILC). IDC and LTM are similar to each other as they both discard samples, but the way the discarded samples are chosen is fundamentally different. As IDC is not able to generalize to larger datasets we also introduce two automatic methods that are able to scale. ILC essentially extends LTM by not discarding the samples, but instead by correcting the labels. We develop new noisy label methods as many of the methods found in the literature have been developed using synthetically generated noise, which has been found to be different from real world noise [12]. This hinders the methods' ability to perform on real world datasets like the ME datasets. In addition, our task is harder in the sense that we have no clean validation set.

3.3.1. Initial Data Cleaning (IDC)

It was found in [11] that only using clean data (10% of the full data) can lead to a better performance than using the whole dataset, including the noisy samples. Inspired by this we manually go through the dataset to find clean samples similarly to the previous section. From the 442 samples of MEGC2019 we find a total of 240 clean samples and 202 samples that are noisy, have a potentially incorrect label, ambiguous or do not fit the typical characteristics of MEs found in the previous section. The number of noisy samples is so high because we use a high threshold in determining clean samples. However, after experimentation we found that the performance had decreased when only training with the clean samples. A finding in [44] points out that data cleaning methods may be cleaning too many samples, discarding samples that are crucial for the training. In the work [66] the authors find that regularizing the network early significantly prevents the network from memorizing the noisy labels in the later stages of the training. Inspired by this we propose a warm-up period during which the network is only trained with clean labels for some duration of epochs, hence the data cleaning is only done *initially*. After the warm-up period the network has access to all of the data, including the samples with noisy labels, allowing access to potentially important samples.

3.3.2. Loss Thresholding with Moments (LTM)

We introduce loss thresholding with moments here as it is used as a sub method in ILC. We use an alternative way of performing the q-percentile also based on the small-loss trick. One of the downsides of q-percentile is the need for a warm-up period as the small-loss trick does not apply at the beginning of the training when the losses are essentially random. In [11] the authors propose to store the mean μ and standard deviation σ of the losses of the most recent 100 training samples. Then they threshold the values if the loss value is higher than $\mu + 1.5\sigma$. We use the same idea but set the coefficient to be a hyperparameter α, thus the threshold value is then given by

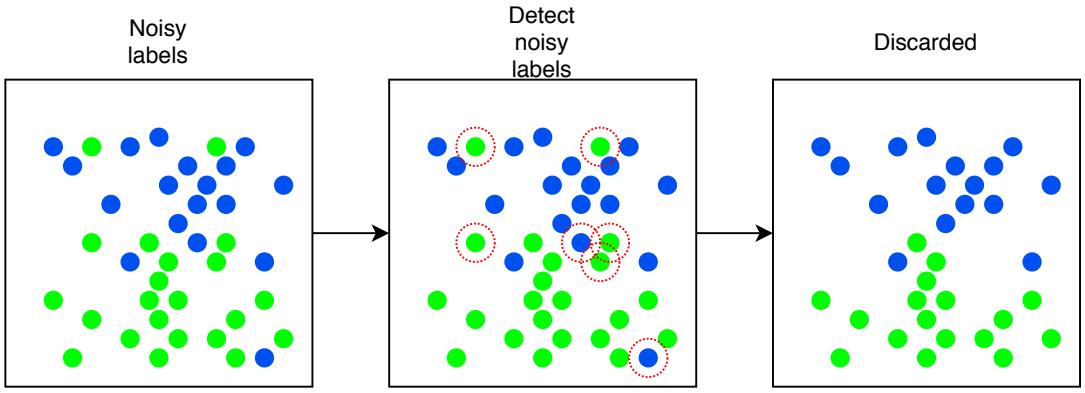


Figure 5. The figure depicts loss thresholding with moments. (Left) We start with a set of noisy labels. (Middle) We detect the noisy labels using the threshold value t given in Equation (4). (Right) The samples that exceed the threshold value t are discarded and we are left with the clean set $\mathcal{C}$ given by Equation (5). LTM leaves us with clean, but fewer data points that are then used to update the model.

$$t = \mu + \alpha\sigma. \tag{4}$$

The hyperparameter α behaves similarly to q and is required as different datasets have different levels of noise. The number of training samples from which the mean and standard deviation are calculated is set to be equal to the number of samples in a batch. This allows us to perform an update based on the most recent loss values and have enough samples to calculate the moments. In addition, by calculating the moments from the batch samples we can avoid more complex implementations. The value may have to be adjusted depending on the size of the dataset and if the ratio of noisy labels is small. We name the method loss thresholding with moments (LTM) as we use the first and second moments of the distribution to formulate the threshold value.

The set of clean labels

$$\mathcal{C} = \{(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y}) \mid \mathcal{L}(f(\mathbf{x}; \boldsymbol{\theta}), y) < t\} \tag{5}$$

is then obtained by thresholding the loss values obtained from an arbitrary loss function $\mathcal{L}$ of each sample, where the network f gives the prediction. Then, the samples with high loss values are ignored when updating the parameters of the network $\boldsymbol{\theta}$ and only the set of clean samples is used in the update with a learning rate of γ. The update for the parameters is then given by

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \gamma \nabla \left(\frac{1}{\mid \mathcal{C} \mid} \sum_{(\mathbf{x}, y) \in \mathcal{C}} \mathcal{L}(f(\mathbf{x}; \boldsymbol{\theta}), y) \right).$$

The benefit of using LTM over q-percentile is that there is no need for a warm-up period at the start of the training. As the threshold value t is given as a function of the moments of the loss distribution, the number of discarded samples is adaptive. Later in Section 5 we show that LTM does not require a warm-up period to work. The method belongs to the class of data cleaning and the process is shown in Figure 5.

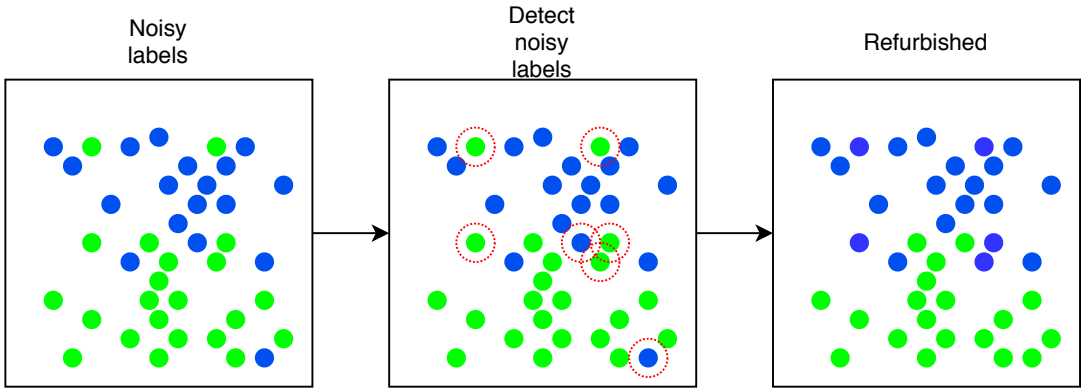


Figure 6. The figure depicts Iterative label correction. (Left) We start with a set of noisy labels. (Middle) We detect the noisy labels similarly to LTM. (Right) The samples that are detected as noisy labels are refurbished by the network's most recent prediction and we are left with the clean set $\mathcal{C}$ and the corrected set $\mathcal{R}$ given by Equation (6). By correcting the labels we are left with the same amount of samples that we started with.

3.3.3. Iterative Label Correction (ILC)

An issue with LTM, and overall with methods discarding samples, is that the discarded samples may be important for the training in small datasets. Therefore we would like to use a label refurbishing method. As discussed in Section 3.1.3, SELFIE uses the idea that not all samples have to corrected, but only the ones we are uncertain of. Our method is similar to SELFIE, but it differs in a few points. Firstly, for the base method of choosing the high loss samples which will be refurbished, we use LTM, *i.e.*, Equation (5), while SELFIE uses q-percentile. Secondly, SELFIE also corrects some of the labels of samples that had been confirmed clean by the cleaning process, while we only correct the label if it is considered to be a high loss sample. The corrected label is determined similarly but we simplify and use the current prediction of the network compared to the most common prediction over some period of time used by SELFIE, as the network could have recently learned something new, making the most recent prediction the most reliable. The set of corrected labels

$$\mathcal{R} = \{(\mathbf{x}, \hat{y}) \mid (\mathbf{x}, y) \in \mathcal{C}^c,\ \hat{y} = f(\mathbf{x}; \boldsymbol{\theta})\} \tag{6}$$

is then given by the predictions of the network for samples that were not found to be clean, *i.e.,*, the complement of $\mathcal{C}$. The parameters are then updated using both the clean set $\mathcal{C}$ and the corrected set $\mathcal{R}$ by

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \gamma \nabla \left(\frac{1}{\mid \mathcal{C} \cup \mathcal{R} \mid} \sum_{(\mathbf{x},y)\in\mathcal{C}} \mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}), y) + \sum_{(\mathbf{x},\hat{y})\in\mathcal{R}} \mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}), \hat{y}) \right).$$

Like SELFIE, ILC also requires a warm-up period as the predictions from the network would not be reliable at the beginning of the training. Algorithm 1 describes the process of ILC for a single epoch and Figure 6 illustrates the method.

The correction of labels can be risky however. If the network has not learned the dataset correctly it is not able to distinguish between the samples and the corrected labels may be incorrect. Giving the high loss samples incorrect labels could lead to an

increase in the number of noisy labels. One of the bigger benefits of using LTM instead of q-percentile is that we can start discarding the samples already in the warm-up phase as seen in Algorithm 1. This mitigates the possibility that the network will overfit to the noisy samples later in the training [66].

Algorithm 1. Iterative label correction

Input : A dataset $\mathcal{D}$, noise level α, warm-up γ, current epoch i, neural network f.

#Draw a batch
for $(\mathbf{X}, \mathbf{y}) \in \mathcal{D}$ **do**
  $\hat{\mathbf{y}} = f(\mathbf{X}; \boldsymbol{\theta})$ #Network's prediction
  $\boldsymbol{\ell} = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ #Vector of losses
  $t = \mu + \alpha\sigma$ #Use $\boldsymbol{\ell}$ to calculate the threshold value by Equation (4)
  $\mathcal{C} = \{(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y}) \mid \boldsymbol{\ell} < t\}$ #Set of samples with clean labels
  if $i < \gamma$ **then**
    #Warm-up
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \gamma\nabla\left(\frac{1}{|\mathcal{C}|}\sum_{(\mathbf{x},y)\in\mathcal{C}}\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}), y)\right)$ #Update parameters only from clean samples
  end
  else
    $\mathcal{R} = \{(\mathbf{x}, \hat{y}) \mid (\mathbf{x}, y) \in \mathcal{C}^c,\ \hat{y} = f(\mathbf{x}; \boldsymbol{\theta})\}$ #Set of noisy samples with corrected labels
    #Update parameters from both clean and corrected samples
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \gamma\nabla\left(\frac{1}{|\mathcal{C}\cup\mathcal{R}|}\sum_{(\mathbf{x},y)\in\mathcal{C}}\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}), y) + \sum_{(\mathbf{x},\hat{y})\in\mathcal{R}}\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}), \hat{y})\right)$
  end
end

4. EXPERIMENTAL SETTINGS

This chapter covers everything related to running the experiments: models, methods, evaluation techniques and metrics. The results themselves are left to the next chapter. We set a single seed for all of our experiments to avoid running multiple models multiple times. We confirmed that the seed used is a representative sample by running five runs with different seeds and averaging the result. We did this for two models and found the absolute difference and standard error to be 0.0012 ± 0.00049 (STSTNet) and 0.0013 ± 0.00042 (SSSNet24) in $F1\text{-}score$ between the five runs averaged together and the run using the set seed.

4.1. Used Models and Settings

For experimentation we use a few models from the literature of MEs. We will use these methods to further evaluate the reliability of our suggested methods. These models are the STSTNet [32], Off-ApexNet [39] and RCNs [40]. The three models were already introduced in Section 2.3.3. In addition we will introduce a simple FCNet (fully connected network) for a baseline and our own model, SSSNet (shallow single stream network), inspired by both STSTNet and Off-ApexNet. All the models use the base settings described below unless otherwise stated specifically for the model. The number of epochs is set to 200; the optimizer used is SGD (stochastic gradient descent) with a learning rate of 0.01, a momentum of 0.9 and a weight decay of 0.001; the dropout is set to 0.5 and the batch size is 128. No data augmentation or normalizing was performed. The input size is set to 60×60 based on findings from [40] and the input includes all of the three components of the apex frame OF $[V_x, V_y, \epsilon]$.

Off-ApexNet

We recreate the model based on the authors implementation using Pytorch [67]. The input size is 28×28 with only the two OF components $[V_x, V_y]$. We were unable to get similar results to theirs using their settings, however by changing the number of epochs to 200 and learning rate to 0.0005 we were able to achieve very similar results, which validates our implementation. This maybe due to using a different method for calculating the OF.

STSTNet

STSTNet is also re-implemented in Pytorch. The input size is also 28×28. For STSTNet we found similar problems with the performance as [40] and the results are therefore different as compared to their original results. Here we used 500 epochs with a learning rate of 0.0005. This may also be due to using a different OF.

RCN

Since the RCNs include multiple different versions, we simply chose one: the attention module RCN-A as it performed the best. We use the code provided by the authors,

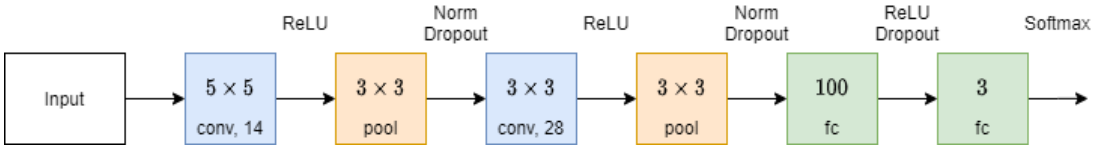


Figure 7. Network architecture of the SSSNet.

but change the $F1\text{-}score$ to match the one used in MEGC2019. We follow the experimental settings from their implementation.

FCNet (fully connected network)

We use one of the simplest possible NNs for a baseline and as a comparison between more complex models. This network consists of just the input layer and the output layer, with zero hidden layers. In addition it contains a ReLU (rectified linear unit) [32] to make the network nonlinear and a dropout for regularization.

SSSNet (shallow single stream network)

Based on the deep learning methods we propose our own network that is inspired by Off-ApexNet and STSTNet. The network consists of only one stream to allow the hidden layers to take combinations of all the channels, unlike in STSTNet and Off-ApexNet, where each input channel has its own stream. Based on the observation from [40] that shallow NNs with fewer parameters seem to perform better for ME recognition, we also propose a network with low number of parameters and only a few layers. The network architecture can be seen from Figure 7. Similarly to Off-ApexNet we have two convolutional layers that are accompanied by a ReLU, pooling, batch norm and a dropout, but we only keep two fully connected layers to avoid a large model. The number of hidden channels was initially set to 14 for the first layer and 28 for the second based on Off-ApexNet. After some experimentation we found 24 channels in the second channel can perform better. We refer to the first one as SSSNet28 and SSSNet24 for the second network.

4.2. Experimental Settings for the Noisy Label Methods

We present the hyperparameters used in the experiments for different methods here. All of the methods were introduced in Section 3.1. For generalized cross-entropy we use a $k = 0.5$, which equals to having half of MAE and half of cross-entropy. For T-revision we estimate the transition matrix using 20 epochs, the actual training is done with 200 epochs and the revision phase is done also in 200 epochs. In Manifold Mixup we use the $\alpha = 0.2$. In q-percentile we found the value of $q = 10$ to be a good choice, indicating that 10% of the data would be noisy. For the warm-up phase $\frac{1}{4}th$ of the epoch was found to be the good after some experimentation. The same value of $\frac{1}{4}th$ was found to be good for IDC. In LTM we use an α of 2. For ILC the warm-up was found to be different compared to q-percentile and LTM, as it requires the network's predictions to be reliable. The value is $\frac{1}{2}$ of the training epochs.

4.3. Evaluation

Evaluation of ME recognition is slightly trickier than the typical train-test split validation using accuracy employed frequently in machine learning. Due to facial-expressions being subjective, a subject independent evaluation is required. In addition, due to the imbalanced setting of the emotions a more well-suited metric that takes into account the imbalance is required.

Metrics

As the emotion classes are imbalanced , $F1\text{-}score$ is used to better take into account classes with smaller number of samples. As there are many different variations of the $F1\text{-}score$ one needs to be careful to use the correct one. The suggested metric from MEGC2019 [20] is the

$$F1\text{-}score_{macro} = \frac{1}{N}(F1_{class_1} + F1_{class_2} + \cdots + F1_{class_N}), \tag{7}$$

also known as $F1\text{-}score_{unweighted}$. The class specific $F1\text{-}score$ can be calculated as

$$F1_{class} = \frac{2TP_{class}}{2TP_{class} + FP_{class} + FN_{class}},$$

where TP refers to true positives, FP to false positives and FN to false negatives. We simplify the notation from $F1\text{-}score_{macro}$ to $F1\text{-}score$ ($F1$ for short) from this point on. One has to be careful when calculating the $F1\text{-}score$ in a cross validation. All of the components (TP, FP, FN) should first be calculated for all the samples, only after which the $F1\text{-}score$ is calculated. Calculating the $F1\text{-}score$ separately for each subject and then averaging the result significantly underestimates the performance of the method [68]. Also, using the alternative way of calculating the $F1\text{-}score$ by using the precision and recall can also significantly underestimate the result.

In addition to $F1$, accuracy is also used. However, due to the imbalanced problem, a modified version, UAR (unweighted average recall) also known as the balanced accuracy is used instead. Similar to $F1$ it takes into account the per-class accuracy by averaging over the classes as

$$\text{UAR} = \frac{1}{N}\sum_{class_n}\frac{TP_{class_n}}{n_{class_n}}, \tag{8}$$

where n_{class_n} is the number of samples for the nth class.

Validation technique

Due to emotions being subjective, typical validation techniques such as train-test-split or k-fold cross validation should not be used, as the dataset contains multiple samples from the same subject. Therefore, the Leave-One-Subject-Out (LOSO) protocol is applied. LOSO splits the data to a training and a test set k times, where k is the number of subjects. In each split, the test set contains one subject's all samples and the training set contains all the samples where the subject is not present. The splits are then repeated for each subject, to ensure a fair evaluation.

5. RESULTS

This chapter presents the results of our experiments. We first start by comparing different noisy label methods, then we look at the generalization ability of our proposed methods over different networks, after which an in-depth analysis of the proposed methods is performed by inspecting the training process, then we compare our methods to the state of the arts and finally perform further experiments and discuss the results. As mentioned in the previous chapter the experiments will be conducted on MEGC2019 with the input being the OF of the apex frame for most methods. The metrics used are the $F1\text{-}score$ from Equation (7) and UAR from Equation (8), with the validation technique being LOSO. For some experiments we deviate from the MEGC2019 standard of presenting the results for all sub datasets and the UAR to save space.

5.1. Comparison of Noisy Labels Methods

We find representative methods from the different groups of noisy label methods presented in Chapter 3 and experiment with them. We use $q - percentile$ [11], Manifold Mixup [52], Generalized cross-entropy [46], T-revision [59] and Meta-Weight-Net [60].

Experiments with different noisy label methods

Table 4 shows the results from different noisy label methods. We show the best method in bold and the second best with an underline. In the experiments we have used SSSNet24 for the baseline, as well as a backbone for the other methods. In the middle part of the table we showcase the results using methods from the literature. All of the methods are able to outperform the baseline for MEGC2019 except for T-revision. We hypothesize that since the matrix transition methods model the transition of classes, it is better suited for methods with a high number of noisy labels and larger datasets. In addition, the transition matrix only mitigates the effects of noisy labels, but as we will show in Section 5.2 the data also contains samples with noise in the data itself, which the other methods are also able to detect. The other methods are able to increase the performance of the baseline, but quite moderately. The best method is Meta-Weight-Net, which is also able to perform the best for the SAMM dataset. Although not performing well on SAMM, the generalized cross-entropy achieves the best result on SMIC and overall a relatively good performance on the MEGC2019. In the bottom section of the table we have the proposed methods. These methods were found to better perform for ME recognition. The initial data cleaning method is able to increase the baseline by 4%, while the loss thresholding with moments and iterative label cleaning fall a bit short of that. The success of multiple different noisy label methods indicates the presence of noisy labels in the dataset. The relative incremental increases in performance can be explained as some of the methods are designed for larger datasets with higher amounts of noisy labels. In addition, a recent work from [12] discusses the differences between real world label noise and synthetically generated label noise and find the distributions to be relatively different. And since most of the noisy label

Table 4. $F1$-$scores$ from different noisy label methods. The baseline and backbone for all the methods is SSSNet24

Model	$F1$			
	MEGC2019	SMIC	CASME II	SAMM
Baseline	0.7415	<u>0.7005</u>	0.8739	0.6302
q-percentile	0.7489	0.6792	0.8844	0.6601
Manifold Mixup	0.7472	0.6901	0.8686	0.6639
Generalized cross-entropy	0.7507	**0.7103**	0.8796	0.6341
T-revision	0.7322	0.6574	0.8389	0.6699
Meta-Weight-Net	0.7527	0.6861	0.8442	**0.7008**
LTM	0.7553	0.6895	0.8838	0.6738
ILC	<u>0.7639</u>	0.6919	<u>0.8995</u>	0.6829
IDC	**0.7707**	0.6979	**0.8996**	<u>0.6919</u>

methods in the literature have been developed using synthetically generated noise, they may fall short with real world data. Probably the biggest reason as to why the improvements are only incremental compared to what the noisy label methods are able to achieve in their own benchmarks, is that we do not have access to clean validation data. Due to the constraints of the LOSO protocol, all of the samples have to be tested. This means that even if we were able to perfectly detect all of the noisy samples, the improvements will be marginal as the testing data contains noisy labels. However, the improvements achieved here are still a sign that the methods are able to better generalize and do not overfit to noisy labels as heavily.

Experiments for architecture generalization

We further experiment with the proposed methods to see if they are able to generalize over different architectures used in the literature of MEs. The results are showcased in Table 5. LTM increases the performance in most cases, except for Off-ApexNet and RCN-A. We are not entirely sure for the reason, but hypothesize it may have something to do with the size of the models as both the models are relatively large. We compare ILC to LTM as the thresholding is a part of the correction process. For all of the cases except for our networks and RCN-A the performance is decreased compared to LTM. This is most likely due to the network's predictions being wrong for weaker networks causing the method to create more noisy labels. ILC requires the network to be well calibrated and to be able to predict the samples with high confidentiality. We show more analysis on this in the next section. IDC is able to increase the performance in every method except for the Off-ApexNet and RCN-A, similarly to LTM. The performance is increased by 4% in SSSNet24 and in STSTNet. The results are somewhat inconsistent between the different networks, but still mostly show the effectiveness of noisy label methods for ME recognition. We also make an interesting observation with FCNet, although the model is almost as simple as it can get, it is still able to achieve a baseline similar to the more complex models. This further suggests that the model's architecture does not appear to be as import for ME recognition.

Table 5. $F1\text{-}scores$ from our proposed noisy label methods applied to various architectures

Model	MEGC2019			
	Baseline	LTM	ILC	IDC
FCNet	0.6814	0.6900	0.6815	0.7050
Off-ApexNet	0.7185	0.7074	0.6971	0.7184
STSTNet	0.7095	0.7348	0.7236	0.7348
RCN-A	0.7168	0.7149	0.7149	0.7117
SSSNet28	0.7362	0.7518	0.7527	0.7468
SSSNet24	**0.7406**	**0.7553**	**0.7639**	**0.7707**

5.2. Analysis of Results

We further analyze the results shown in the previous section. We first show the difference between q-percentile and loss thresholding with moments by analyzing the change of loss distribution and the threshold values throughout the training. We show the effectiveness of LTM in finding potentially noisy labels and the effectiveness of ILC in refurbishing the potential noisy labels by using a two dimensional projection of the hidden space to qualitatively analyze the methods.

Q-percentile versus loss thresholding with moments

The biggest difference between the two methods is that the threshold percentage is adaptive in LTM. Where q-percentile always keeps the q static through the training the "q" (*i.e.*, the percentage of discarded samples) in LTM adapts the value throughout the training epochs. Figure 8 showcases the evolution of the threshold values over the training. The losses follow the normal distribution at the start of the training as the predictions are done randomly—by the end of the training the samples follow a distribution that resembles a beta distribution, but with a heavy tail due to the noisy labels. As the predictions are done randomly at the start of the training and are only slowly becoming more reliable, the loss values at the start of the training are not a reliable metric and the small-loss trick does not apply. Loss thresholding with moments is able to automatically warm-up due to using statistics from the distribution, whereas q-percentile requires a hyperparameter to be set for the warm-up period.

Effectiveness of loss thresholding with moments and iterative label correction

We showcase the effectiveness of LTM and ILC by looking at the samples that were found to be high loss samples. We show the first 15 samples with high loss values in Figure 9. We can see some possibly incorrect labels in samples (0, 4, 9, 12, 13), noisy or ambiguous samples in (1, 3, 6, 7, 8, 11) and possibly false positives that seem to be clean but difficult samples in (2, 5, 10, 14). The LTM shows its effectiveness in finding not only samples with noisy labels but also samples that have noise in the input. One of the downsides of the method is the false positives that seem to be mostly samples that are simply challenging to classify.

Figure 10a shows a projection from the last hidden layer to two dimensions using UMAP (uniform manifold approximation and projection) [69]. The classes cluster

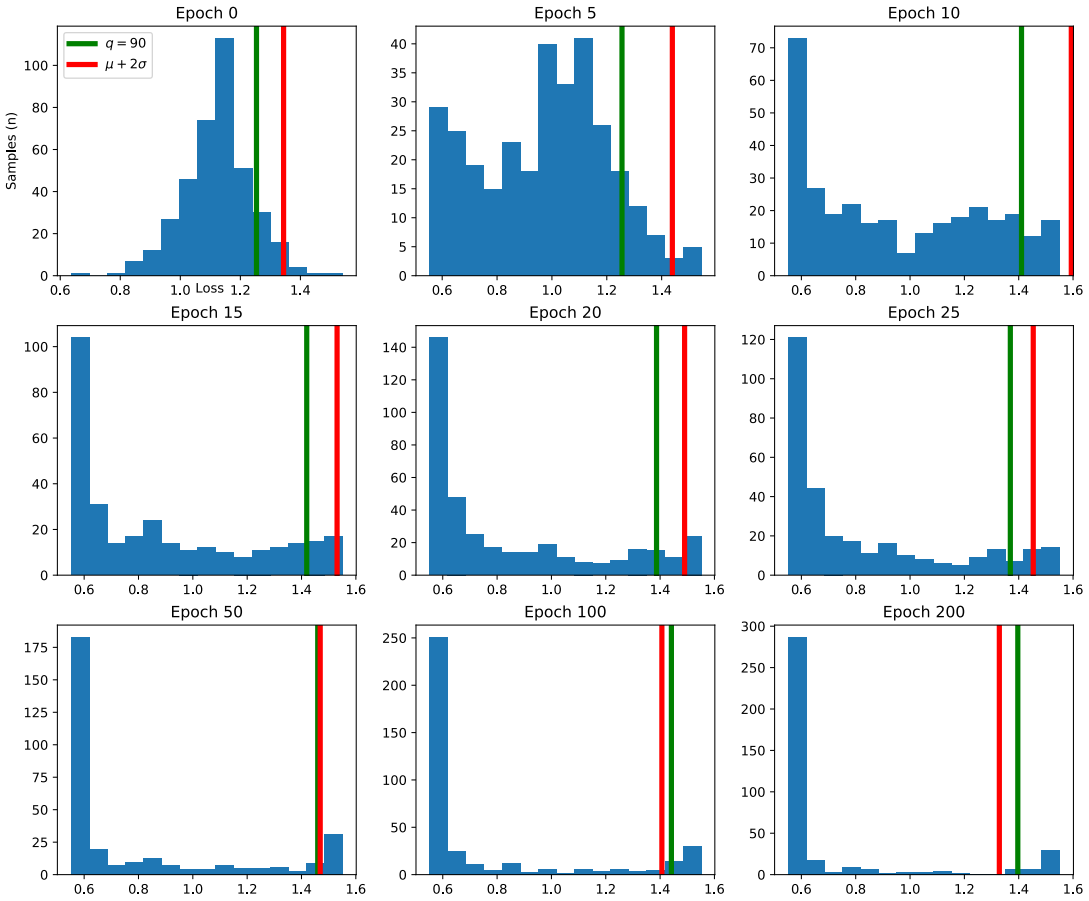


Figure 8. The loss distributions throughout the training. The green line shows the threshold value for q-percentile and the red line shows it for loss thresholding with moments. In both cases, the samples on the right side are considered as the high loss samples. We can see that during the early epochs the q-percentile is discarding many more samples compared to loss thresholding with moments, but at the end of the training the two lines are very close to each other.

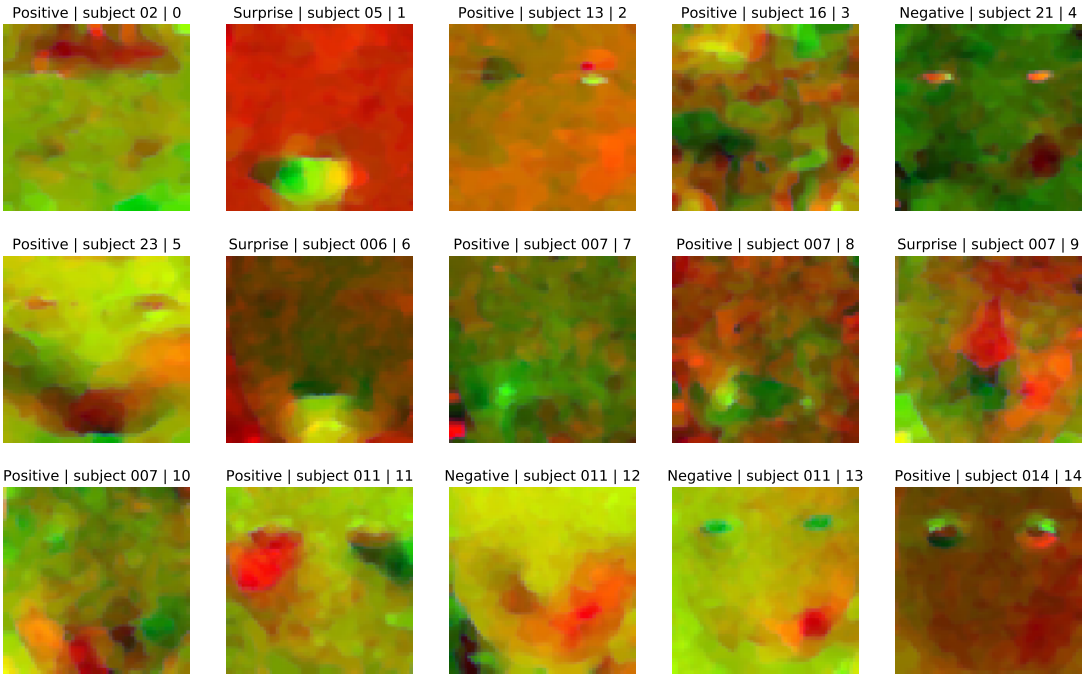
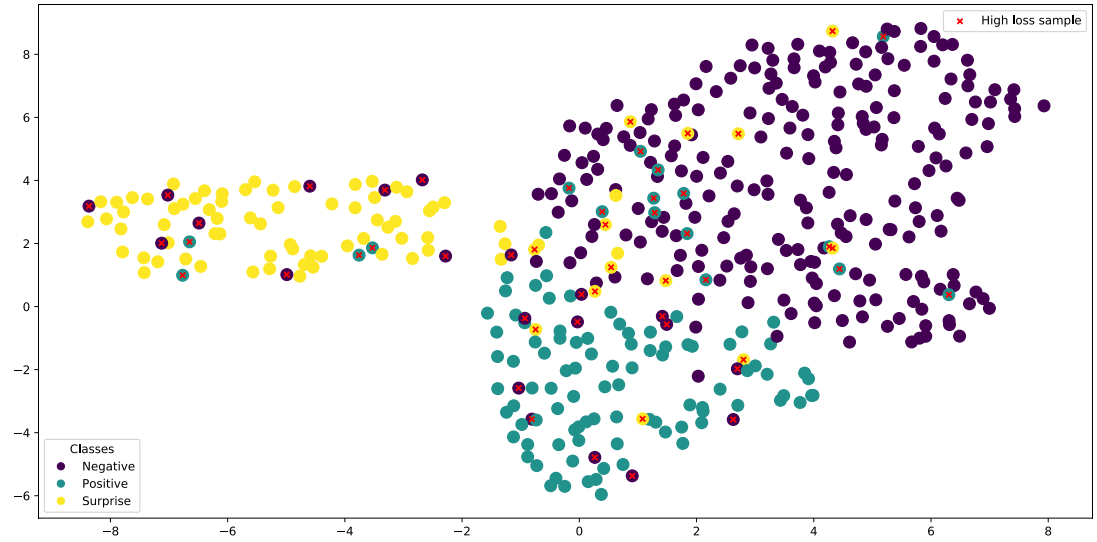


Figure 9. The first 15 samples with a high loss based on LTM. The title of each image consists of *the emotion class* | *the subject number* | *the sample number from the high loss samples*.

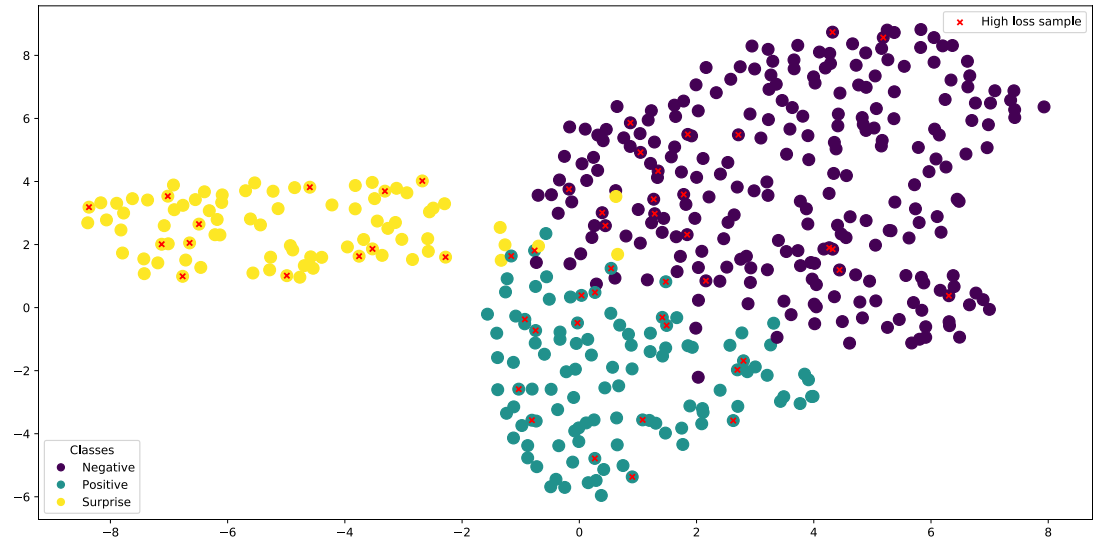
relatively well and the class borders can be roughly drawn. Some samples seem to be in the wrong clusters, which we have been able to detect using loss thresholding with moments. Unfortunately, not all samples that seem to be in the wrong clusters have been detected, the two *surprise* samples at the border of *negative* and *positive*. In addition, samples with a correct label may also have been marked as high loss samples as can be seen with samples marked at the classification border of *negative* and *positive*.

Figure 10b shows the same projection, but the labels from the samples that were detected as high loss samples have now been corrected with ILC. Here we can see that essentially all of the samples were corrected to a label that best represents their neighboring points. Once more, some arguments can be made about the samples in the classification border of *negative* and *positive*.

Now we will examine the possible explanation for the ILC's inability to generalize over different architectures. Figure 11 shows the UMAP projections for STSTNet. We can immediately observe that the classification borders are nearly not as well separated as in Figure 10. This makes finding the high loss samples harder and refurbishing them can lead to mistakes as the predictions can be wrong, as can be seen from Figure 11b. For ILC to work properly, the network has to be able to separate the samples well enough such that the predictions are reliable. For SSSNet24 in Figure 10 the classification borders are much more visible, which makes the predictions for the network more reliable.

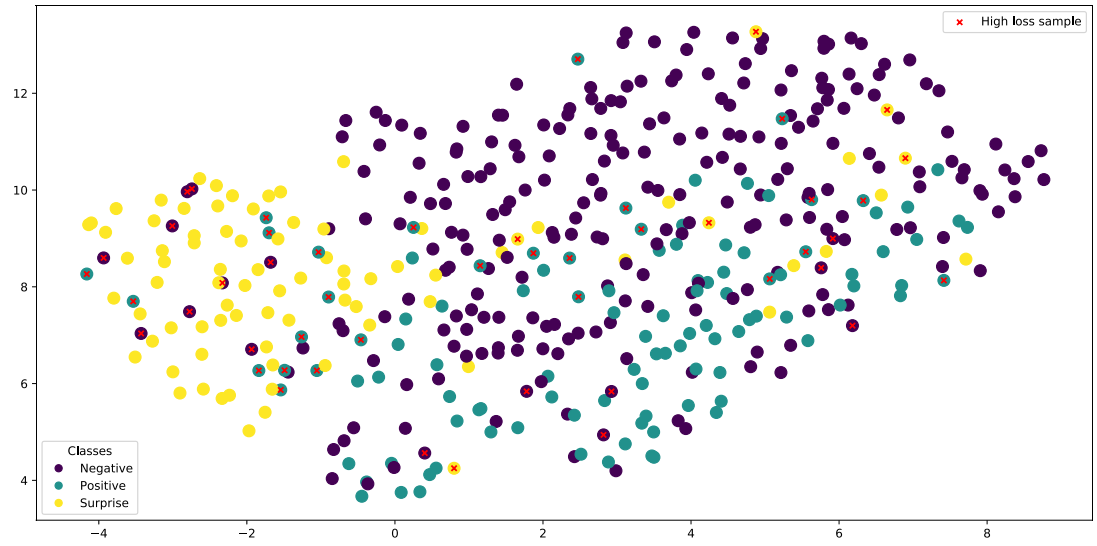


(a) A UMAP projection to two dimensions from the last hidden layer. Each sample has been colored by their respective class. The red crosses mark samples that exceed the threshold and are thus determined to be high loss samples.

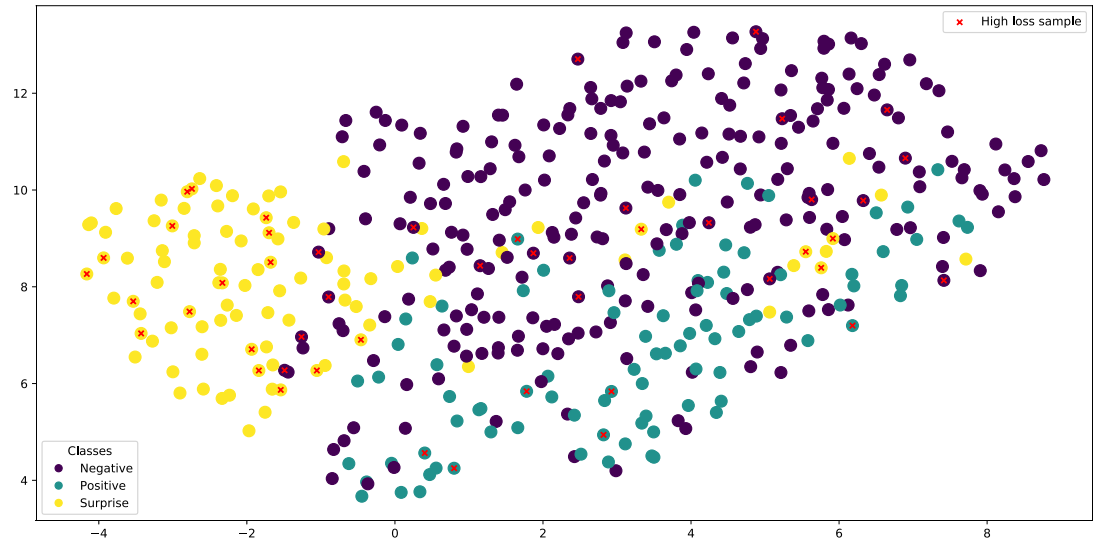


(b) The same as above, but the high loss samples have been refurbished using ILC.

Figure 10. UMAP projections from the last hidden layer of SSSNet24.



(a) Detected high loss samples



(b) Refurbished high loss samples

Figure 11. UMAP projections from the last hidden layer of STSTNet.

Table 6. Comparison to the state-of-the-art methods

Model	MEGC2019		SMIC		CASME II		SAMM	
	UAR	$F1$	UAR	$F1$	UAR	$F1$	UAR	$F1$
LBP-TOP [26] [2]	0.5785	0.5882	0.5280	0.2000	0.7429	0.7026	0.4102	0.3954
MDMO [21]	0.5782	0.5881	0.5511	0.5587	0.7925	0.8014	0.3073	0.3065
Off-ApexNet [39]	0.7164	0.7176	0.6972	0.7039	0.8094	0.8159	0.6172	0.6126
STSTNet [32]	0.7125	0.7095	0.6726	0.6618	0.8132	0.8325	0.6257	0.6176
NMER [41] [3]	0.5936	0.5936	0.5555	0.5607	0.6929	0.7624	0.4894	0.6389
RCN-A [40]	0.7138	0.7168	0.6619	0.6590	0.7894	0.8109	0.6531	0.6547
SSSNet24	0.7505	0.7415	<u>0.7071</u>	**0.7005**	0.8739	0.8716	0.6417	0.6302
SSSNet24 + LTM	0.7647	0.7553	0.7009	0.6895	0.8814	0.8838	0.6848	0.6738
SSSNet24 + ILC	<u>0.7729</u>	<u>0.7639</u>	0.6979	0.6919	<u>0.9051</u>	<u>0.8995</u>	<u>0.6919</u>	<u>0.6829</u>
SSSNet24 + IDC	**0.7794**	**0.7707**	**0.7090**	<u>0.6979</u>	**0.9060**	**0.8996**	**0.6990**	**0.6919**

5.3. Comparison to State of the Arts

We compare our methods to the state-of-the-art methods from the literature in Table 6. We reimplement all the methods except for LBP-TOP and NMER. The source for results of LBP-TOP and NMER can be found in the footnote. At the top we have the two traditional methods LBP-TOP and MDMO. Both of the methods achieve similar results at around 0.58 $F1$. In the middle we have the current state-of-the-art methods for ME recognition that have been evaluated using the MEGC2019 dataset. All of the methods were recreated by us except the NMER. The methods in the middle are all neural network models, which shows the effectiveness of deep learning. Our network with the proposed methods are presented in the bottom of the table. The IDC method is able to achieve the state of the art for ME recognition of 0.77 in the $F1\text{-}score$. Both the ILC and IDC are very close to breaking the 0.90 mark for $F1$ in the CASME II dataset. Significant increases can be seen in both CASME II and SAMM compared to the previous state-of-the-art methods.

5.4. Further Experiments

We show further experiments for cross-dataset evaluation as well as for neural architecture search. We perform cross-dataset evaluation to see if our methods are able to create domain-independent features and generalize over different datasets. Neural architecture search was initially performed in an attempt to find a good network, but without improvements. We however find a possible link between the NAS results and the cross-dataset evaluation.

Cross-dataset evaluation

In the cross-dataset task we evaluate our method by training on datasets that are different from the dataset used in the testing. This is essentially the Leave-One-

[2] The results are from [20].

[3] The results are from [40], where the method was recreated.

Subject-Out (LOSO), except it is done for the databases so we get the Leave-One-Dataset-Out protocol. In our experiments we first train the network using two datasets from the MEGC2019 and then test on the one that is left out. In the table we use the abbreviations CA for CASME II, SM for SMIC and SA for SAMM.

Table 7 shows the results for the cross-dataset task. The baselines, where all of the datasets are used, are shown in Table 6. We use multiple models to validate our results. For SMIC the performance decreases for SSSNet24, but for RCN-A and STSTNet it stays the same. The results significantly decrease in CASME II, which indicates that the dataset contains important samples for the training of the network. For SAMM, the results show something else. The performance for SSSNet24 seems to increase as we remove the SAMM dataset. We are not entirely sure why, but hypothesize that the possible reason may have to do something with the fact that SAMM is the hardest dataset out of the tree, and may contain difficult samples that may negatively impact the training. Also, SSSNet24 has a relatively poor performance for SAMM compared to SMIC, it may be that SSSNet24 "overfit" to SMIC during the baseline training.

Table 7. Results for cross-dataset evaluation

Model	CA & SA $\rightarrow$ SM		SM & SA $\rightarrow$ CA		CA & SM $\rightarrow$ SA	
	UAR	$F1$	UAR	$F1$	UAR	$F1$
STSTNet	0.6577	0.6623	0.7115	0.7478	0.6906	0.6171
RCN-A	0.6512	0.6522	0.7100	0.7182	0.6828	0.6548
SSSNet24	0.6554	0.6581	0.8273	0.8508	0.6630	0.6525

Neural architecture search

We initially employed a neural architecture search (NAS) in hopes to find a better architecture for the task. We used AutoSlim [70], a one-shot neural architecture search that uses Slimmable Networks [71] to search for the number of hidden features. The search is performed by switching the number of hidden features in each layer for every new iteration (when a new minibatch is drawn). We set the number of searchable features for the first and second convolutional layers and the fully connected layer in SSSNet. For the first layer we use the the set $[6, 24]$, for the second $[16, 42]$ and for the fully connected the set $\{25, 50, 75, 100, 150, 200, 250, 300, 400\}$. In the validation set that is used to evaluate which set of hyperparameters performs the best, we only include subjects that are not in the training set and experimented with different ratios of training and validation set sizes.

Table 8 showcases the results for the NAS. The results are overall worse than SSSNet24 and SSSNet28 shown in Tables 5 and 6. We hypothesize that the reason for the inability to find good architectures is due to the need of a validation set, as the domain shift between the test set and validation set is too high.

Table 8. Results for neural architecture search

Model	MEGC2019		SMIC		CASME II		SAMM	
	UAR	$F1$	UAR	$F1$	UAR	$F1$	UAR	$F1$
SSSNet + AutoSlim	0.7505	0.7352	0.6787	0.6818	0.8663	0.8590	0.6725	0.6529

5.5. Discussion

Noisy labels in ME recognition

One of the major differences to the experiments performed in the literature for noisy label methods is that we do not have a clean validation set. Due to the constraint of subjectivity of emotions the LOSO protocol is required. This means that even if we were able to find the samples with noisy labels and then be able to refurbish them perfectly, the performance would not change drastically as the test data would also include those samples with noisy labels. However, the noisy label methods still seem to be able create more generalizable features that are capable of increasing the performance.

It would be interesting to know the ratio of noisy labels in the MEGC2019 dataset to better understand the scale of the problem. In initial data cleaning, we chose 240 samples that were clean and 202 that were noisy. From those 202 we would say about 50-70 seem to be samples with noisy labels. With q-percentile we found the best q to be 10, and with loss thresholding with moments $\alpha = 2$, corresponded to 42 samples at the end of training. From the different methods, it would seem that about $10 - 15\%$ of the samples would contain noisy labels. This would correspond to the reliability score of labelling AUs in SAMM and CASME II, where the score was around 82%. However, due to the unavailability of clean validation data these estimates for the ratio of noisy labels in MEGC2019 are unreliable.

Limitations

The shown results give an indicator towards the existence of noisy labels in the micro-expression datasets. However, due to the complexity of MEs we cannot be certain of this. Our work is limited in the sense that we only observe the OF domain. The reason to focusing only on the OF domain is because it is easier to observer and there are currently no works that have successfully employed methods based on the original input domain. To confirm the incorrect labels, the data would have to be looked in the original domain by a professional, and even then due to the subjectivity of emotions we could not be certain. The possible noisy labels we observed in this thesis may also just be an artifact of observing an aggregated version of two frames. By looking outside the apex in the original domain we could find that the labels are correct. Nonetheless, whether the dataset contains noisy labels or not, the methods used appear to be effective in increasing the performance. A future work could however be looking at the noisy label problem more carefully from the videos.

Future datasets

For future ME datasets it could be beneficial to include labels from all the annotators rather than an aggregated version, as [58] showed that it is possible to increase the performance by modelling the annotator's individual transition matrices. Another idea is to let the annotators choose multiple labels for a single sample if it depicts multiple emotions and (or) have a confidentiality measure for each label. Including more information about the labelling process could help compared to just aggregating the information.

Domain shift

One of the major problems in addition to label noise we found is domain shift. We lightly touched on this earlier in Section 2.2.4, where we mentioned that the combination of different datasets for MEGC2019 creates a domain shift between the datasets. The experiments in Section 5.4 show only a moderate issue in domain shift between the different datasets in MEGC2019, and even an opportunity as we saw an increase in the performance for SSSNet24. An even bigger problem seems to be the domain shift between the subjects themselves. This domain shift is a problem in itself, as methods can learn domain dependent features that can be harmful for generalization. An even bigger problem occurs when using methods that rely on optimizing over validation data. We first observed this problem when attempting to use early stopping to find the optimal number of epochs, however we could only make the performance worse. We also experimented with NAS, which requires the use of validation set to find the optimal hyperparameters. NAS has great potential, especially in the LOSO protocol, as we could potentially find a personalized architecture for each subject. However, the requirement of using a validation set seems to hinder the viability, as seen in Table 8. The idea of NAS was also experimented in [40], but with similar results. These results give some indication over a domain shift between the subjects themselves. One possible solution for the problem is to use domain generalization techniques [72]. The main strategy of these methods is to create domain invariant features that are usable throughout the dataset independent of the domains. We leave these sets of methods as a future work.

6. CONCLUSION

This thesis showcased the effectiveness of using noisy label methods on micro-expression recognition. We started off by observing the labelling process of the datasets and by looking at individual samples, in order to better understand the problem of noisy labels. We performed an exhaustive literature study of different noisy label methods and proposed two methods: initial data cleaning and iterative label correction. In initial data cleaning we manually go through the dataset to find clean samples and only use the clean samples in the initial stages of the training. Iterative label correction finds noisy samples using loss thresholding with moments and then corrects the noisy labels using the network's predictions. We perform extensive experiments using noisy label methods for micro-expression recognition and find that nearly all of the methods are able to increase the performance, indicating the existence of noisy samples in micro-expressions datasets. We achieve the state of the art for MEGC2019 with an $F1\text{-}score$ of 0.77 by using the initial data cleaning method showing its effectiveness.

7. REFERENCES

[1] Ekman P. & Friesen W.V. (1969) Nonverbal leakage and clues to deception. Psychiatry 32, pp. 88–106. URL: https://doi.org/10.1080/00332747.1969.11023575, pMID: 27785970.

[2] Oh Y., See J., Ngo A.C.L., Phan R.C. & Baskaran V.M. (2018) A survey of automatic facial micro-expression analysis: Databases, methods and challenges. CoRR abs/1806.05781. URL: http://arxiv.org/abs/1806.05781.

[3] Yan W.J., Li X., Wang S.J., Zhao G., Liu Y.J., Chen Y.H. & Fu X. (2014) Casme ii: An improved spontaneous micro-expression database and the baseline evaluation. PLOS ONE 9, pp. 1–8. URL: https://doi.org/10.1371/journal.pone.0086041.

[4] Davison A.K., Lansley C., Costen N., Tan K. & Yap M.H. (2018) Samm: A spontaneous micro-facial movement dataset. IEEE Transactions on Affective Computing 9, pp. 116–129.

[5] Gan Y., Liong S.T., Yau W.C., Huang Y.C. & Tan L.K. (2019) Off-apexnet on micro-expression recognition system. Signal Processing: Image Communication 74, pp. 129–139.

[6] Liong S.T., See J., Wong K. & Phan R.C.W. (2018) Less is more: Micro-expression recognition from video using apex frame. Signal Processing: Image Communication 62, pp. 82–92.

[7] Zhang C., Bengio S., Hardt M., Recht B. & Vinyals O. (2016), Understanding deep learning requires rethinking generalization.

[8] Goodfellow I., Bengio Y. & Courville A. (2016) Deep Learning. MIT Press. http://www.deeplearningbook.org.

[9] Shin W., Ha J.W., Li S., Cho Y., Song H. & Kwon S. (2020), Which strategies matter for noisy label classification? insight into loss and uncertainty.

[10] A curated list of resources for Learning with Noisy Labels. URL: https://github.com/subeeshvasu/Awesome-Learning-with-Label-Noise. Accessed 24.8.2020.

[11] Karimi D., Dou H., Warfield S.K. & Gholipour A. (2019), Deep learning with noisy labels: exploring techniques and remedies in medical image analysis.

[12] Google AI Blog: Understanding deep learning on controlled noisy labels. URL: https://ai.googleblog.com/2020/08/understanding-deep-learning-on.html. Accessed 20.8.2020.

[13] Li X., Pfister T., Huang X., Zhao G. & Pietikäinen M. (2013) A spontaneous micro-expression database: Inducement, collection and baseline. In: 2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), pp. 1–6.

[14] Li X., Hong X., Moilanen A., Huang X., Pfister T., Zhao G. & Pietikäinen M. (2015) Reading hidden emotions: Spontaneous micro-expression spotting and recognition. CoRR abs/1511.00423. URL: http://arxiv.org/abs/1511.00423.

[15] Merghani W., Davison A.K. & Yap M.H. (2018) A review on facial micro-expressions analysis: Datasets, features and metrics. CoRR abs/1805.02397. URL: http://arxiv.org/abs/1805.02397.

[16] Kollias D. & Zafeiriou S. (2018) Aff-wild2: Extending the aff-wild database for affect recognition. CoRR abs/1811.07770. URL: http://arxiv.org/abs/1811.07770.

[17] Wen-Jing Yan, Wu Q., Yong-Jin Liu, Su-Jing Wang & Fu X. (2013) Casme database: A dataset of spontaneous micro-expressions collected from neutralized faces. In: 2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), pp. 1–7.

[18] Qu F., Wang S., Yan W., Li H., Wu S. & Fu X. (2018) Cas(me)^2: A database for spontaneous macro-expression and micro-expression spotting and recognition. IEEE Transactions on Affective Computing 9, pp. 424–436.

[19] Petr Husak Jan Cech J.M. (2017) Spotting facial micro-expressions “in the wild”. In: 22nd Computer Vision Winter Workshop.

[20] See J., Yap M.H., Li J., Hong X. & Wang S.J. (2019) Megc 2019 – the second facial micro-expressions grand challenge. pp. 1–5.

[21] Liu Y., Zhang J., Yan W., Wang S., Zhao G. & Fu X. (2016) A main directional mean optical flow feature for spontaneous micro-expression recognition. IEEE Transactions on Affective Computing 7, pp. 299–310.

[22] Pfister T., Xiaobai Li, Zhao G. & Pietikäinen M. (2011) Recognising spontaneous facial micro-expressions. In: 2011 International Conference on Computer Vision, pp. 1449–1456.

[23] Krizhevsky A. et al. (2009) Learning multiple layers of features from tiny images .

[24] Deng J., Dong W., Socher R., Li L.J., Li K. & Fei-Fei L. (2009) ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09.

[25] Wu Q., Shen X. & Fu X. (2011) The machine knows what you are hiding: An automatic micro-expression recognition system. pp. 152–162.

[26] Zhao G. & Pietikainen M. (2007) Dynamic texture recognition using local binary patterns with an application to facial expressions. IEEE Transactions on Pattern Analysis and Machine Intelligence 29, pp. 915–928.

[27] Wang Y., See J., Phan R.C.W. & Oh Y.H. (2015) Lbp with six intersection points: Reducing redundant information in lbp-top for micro-expression recognition. In: D. Cremers, I. Reid, H. Saito & M.H. Yang (eds.) Computer Vision – ACCV 2014, Springer International Publishing, Cham.

[28] Wang Y., See J., Phan R. & Oh Y.H. (2015) Efficient spatio-temporal local binary patterns for spontaneous facial micro-expression recognition. PloS one 10, p. e0124674.

[29] Guo C., Liang J., Zhan G., Liu Z., Pietikäinen M. & Liu L. (2019) Extended Local Binary Patterns for Efficient and Robust Spontaneous Facial Micro-Expression Recognition. arXiv e-prints , arXiv:1907.09160.

[30] Sun D., Roth S. & Black M.J. (2014) A quantitative analysis of current practices in optical flow estimation and the principles behind them. International Journal of Computer Vision 106, pp. 115–137. URL: https://doi.org/10.1007/s11263-013-0644-x.

[31] Sun D., Roth S. & Black M.J. (2010) Secrets of optical flow estimation and their principles. In: 2010 IEEE computer society conference on computer vision and pattern recognition, IEEE, pp. 2432–2439.

[32] Liong S.T., Gan Y.S., See J., Khor H.Q. & Huang Y.C. (2019), Shallow triple stream three-dimensional cnn (ststnet) for micro-expression recognition. URL: http://dx.doi.org/10.1109/FG.2019.8756567.

[33] Chaudhry R., Ravichandran A., Hager G. & Vidal R. (2009) Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1932–1939.

[34] Liu Y., Li B. & Lai Y. (2018) Sparse mdmo: Learning a discriminative feature for spontaneous micro-expression recognition. IEEE Transactions on Affective Computing , pp. 1–1.

[35] Nayak K., Wang X.S., Ioannidis S., Weinsberg U., Taft N. & Shi E. (2015) Graphsc: Parallel secure computation made easy. In: 2015 IEEE Symposium on Security and Privacy, pp. 377–394.

[36] Liong S.T., See J., Wong K. & Phan R.C.W. (2018) Less is more: Micro-expression recognition from video using apex frame. Signal Processing: Image Communication 62, pp. 82 – 92. URL: http://www.sciencedirect.com/science/article/pii/S0923596517302436.

[37] Arxiv search for micro-expressions. URL: https://arxiv.org/search/?query=%22micro+expression%22+OR+%22micro-expression%22+OR+%22micro+expressions%22&searchtype=all&abstracts=show&order=-announced_date_first&size=50. Accessed 22.7.2019.

[38] Li Y., Huang X. & Zhao G. (2018) Can micro-expression be recognized based on single apex frame? In: 2018 25th IEEE International Conference on Image Processing (ICIP), pp. 3094–3098.

[39] Liong S.T., Gan Y.S., Yau W.C., Huang Y.C. & Ken T.L. (2018), Off-apexnet on micro-expression recognition system.

[40] Xia Z., Peng W., Khor H.Q., Feng X. & Zhao G. (2020), Revealing the invisible with model and data shrinking for composite-database micro-expression recognition.

[41] Liu Y., Du H., Zheng L. & Gedeon T. (2019) A neural micro-expression recognizer. In: 2019 14th IEEE international conference on automatic face & gesture recognition (FG 2019), IEEE, pp. 1–4.

[42] Liong S.T., Gan Y.S., Zheng D., Lic S.M., Xua H.X., Zhang H.Z., Lyu R.K. & Liu K.H. (2019), Evaluation of the spatio-temporal features and gan for micro-expression recognition system.

[43] Chaudhari P., Agrawal H. & Kotecha K. (2019) Data augmentation using mg-gan for improved cancer classification on gene expression data. Soft Computing , pp. 1–11.

[44] Song H., Kim M., Park D. & Lee J.G. (2020), Learning from noisy labels with deep neural networks: A survey.

[45] Ghosh A., Kumar H. & Sastry P.S. (2017), Robust loss functions under label noise for deep neural networks.

[46] Zhang Z. & Sabuncu M.R. (2018), Generalized cross entropy loss for training deep neural networks with noisy labels.

[47] Xue C., Dou Q., Shi X., Chen H. & Heng P.A. (2019), Robust learning at noisy labeled medical images: Applied to skin lesion classification.

[48] Song H., Kim M. & Lee J.G. (2019) Selfie: Refurbishing unclean samples for robust deep learning. In: International Conference on Machine Learning, pp. 5907–5915.

[49] Shen Y. & Sanghavi S. (2018), Learning with bad training data via iterative trimmed loss minimization.

[50] Northcutt C.G., Wu T. & Chuang I.L. (2017), Learning with confident examples: Rank pruning for robust classification with noisy labels.

[51] Zhang H., Cisse M., Dauphin Y.N. & Lopez-Paz D. (2017), mixup: Beyond empirical risk minimization.

[52] Verma V., Lamb A., Beckham C., Najafi A., Mitliagkas I., Courville A., Lopez-Paz D. & Bengio Y. (2018), Manifold mixup: Better representations by interpolating hidden states.

[53] Lee K.H., He X., Zhang L. & Yang L. (2018) Cleannet: Transfer learning for scalable image classifier training with label noise. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5447–5456.

[54] Reed S., Lee H., Anguelov D., Szegedy C., Erhan D. & Rabinovich A. (2014), Training deep neural networks on noisy labels with bootstrapping.

[55] Arazo E., Ortego D., Albert P., O'Connor N.E. & McGuinness K. (2019), Unsupervised label noise modeling and loss correction.

[56] Sukhbaatar S., Bruna J., Paluri M., Bourdev L. & Fergus R. (2014), Training convolutional networks with noisy labels.

[57] Patrini G., Rozza A., Menon A., Nock R. & Qu L. (2016), Making deep neural networks robust to label noise: a loss correction approach.

[58] Tanno R., Saeedi A., Sankaranarayanan S., Alexander D.C. & Silberman N. (2019), Learning from noisy labels by regularized estimation of annotator confusion.

[59] Xia X., Liu T., Wang N., Han B., Gong C., Niu G. & Sugiyama M. (2019) Are anchor points really indispensable in label-noise learning? In: Advances in Neural Information Processing Systems, pp. 6838–6849.

[60] Shu J., Xie Q., Yi L., Zhao Q., Zhou S., Xu Z. & Meng D. (2019), Meta-weight-net: Learning an explicit mapping for sample weighting.

[61] Carlini N., Úlfar Erlingsson & Papernot N. (2019), Distribution density, tails, and outliers in machine learning: Metrics and applications.

[62] Jiang L., Zhou Z., Leung T., Li L.J. & Fei-Fei L. (2018) Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In: International Conference on Machine Learning, pp. 2304–2313.

[63] Guo S., Huang W., Zhang H., Zhuang C., Dong D., Scott M.R. & Huang D. (2018), Curriculumnet: Weakly supervised learning from large-scale web images.

[64] Malach E. & Shalev-Shwartz S. (2017) Decoupling" when to update" from" how to update". In: Advances in Neural Information Processing Systems, pp. 960–970.

[65] Mancini M., Akata Z., Ricci E. & Caputo B. (2020), Towards recognizing unseen categories in unseen domains.

[66] Liu S., Niles-Weed J., Razavian N. & Fernandez-Granda C. (2020), Early-learning regularization prevents memorization of noisy labels.

[67] Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J. & Chintala

S. (2019), Pytorch: An imperative style, high-performance deep learning library. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[68] Forman G. & Scholz M. (2010) Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. Acm Sigkdd Explorations Newsletter 12, pp. 49–57.

[69] McInnes L., Healy J. & Melville J. (2018), Umap: Uniform manifold approximation and projection for dimension reduction.

[70] Yu J. & Huang T. (2019), Autoslim: Towards one-shot architecture search for channel numbers.

[71] Yu J. & Huang T. (2019), Universally slimmable networks and improved training techniques.

[72] Li D., Zhang J., Yang Y., Liu C., Song Y.Z. & Hospedales T.M. (2019), Episodic training for domain generalization.

[73] Oliphant T.E. (2006) A guide to NumPy, vol. 1. Trelgol Publishing USA.

[74] Hunter J.D. (2007) Matplotlib: A 2d graphics environment. Computing in science & engineering 9, pp. 90–95.

[75] McKinney W. et al. (2010) Data structures for statistical computing in python. In: Proceedings of the 9th Python in Science Conference, vol. 445, Austin, TX, vol. 445, pp. 51–56.

[76] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V. et al. (2011) Scikit-learn: Machine learning in python. Journal of machine learning research 12, pp. 2825–2830.