Syndrome detection using kinship verification

Bachelor's Project Thesis

Arianne Meijer - van de Griend
Supervisors: Marco Wiering, Jayne Hehir-Kwa & Hamdi Dibeklioglu

Abstract: Neural developmental disorders are often associated with dysphomic facial features. Currently, dysmorphic features are diagnosed by clinicians observing patients. Computer image analysis can be use to automatically identify these features, but that requires a large dataset for training. This paper investigates the use of a kinship verification classifier to automatically diagnose a rare neural developmental disorder called Koolen-de Vries Syndrome (KdVS). Since it is not possible to build a large enough dataset, this is done using type II errors as a basis for syndrome detection.

First, the UB KinFace dataset was split into different train and test sets such that these subsets contained different properties and can be used as train and test set interchangeably. Then, several SVM classifiers were trained on these constructed subsets and evaluated on three tasks: kinship verification, identification and syndrome detection. The tasks were separated into a genderless and gender-specific version.

Unfortunately, no conclusions can be drawn on the applicability of this approach due to shortcomings in the KdVS dataset. We did obtained evidence of a problem with the UB KinFace dataset. There exists a possibility that the dataset contains father-child pairs where the father is not the biological father of the child. This unreliable labeling can be the cause of poor performance on the kinship verification task.

In the future computer image processing may be able to aid the diagnosis of patients, however significant technical limitations need to be overcome.

1 Introduction

In clinical genetics, clinicians need expert knowledge for correct diagnosis. However, it is impractical, if not impossible, for every clinician to be an expert on every rare syndrome. Thus, it is desirable to design an algorithm that can help clinicians diagnose rare cases of neural developmental disorders.

In this study, an approach is proposed for the automatic diagnosis of a rare syndrome called *Koolen - de Vries Syndrome* (KdVS). KdVS is a genetic disorder characterized by moderate intellectual disability and characteristic facial dimorphism, among other symptoms (Koolen, Pfundt, Linda, Beunders, Veenstra-Knol, Conta, Fortuna, Gillessen-Kaesbach, Dugan, Halbach, et al., 2016). In general, the parents of someone with KdVS do not have the syndrome (Koolen et al., 2016), this is called a *de novo mutation*. KdVS belongs to a set of neural developmental disorders whereby the patients normal development is severely affected, both neurological and physical. Other, more frequently occurring examples include Down Syndrome.

Since KdVS is a rare syndrome, it is not possible to create a large enough dataset that is suitable for training a classifier directly. Instead, it is proposed to use kinship verification to indicate a suspicion of neural developmental disorders. This could then be used by clinicians as a basis for testing someone for KdVS.

This approach is possible because neural developmental disorders are characterized by abnormal brain development. It is possible to detect this using facial analysis, because the development of the brain of a fetus is linked to the development of its head and face (Aldridge, George, Cole, Austin, Takahashi, Duan, and Miles, 2011; Ferry, Steinberg, Webber, FitzPatrick, Ponting, Zisserman, and Nellåker, 2014; Obafemi-Ajayi, Miles, Takahashi, Qi, Aldridge, Zhang, Xin, He, and Duan, 2015). Thus, kinship verification can be used to identify facial dimorphism by looking at false negatives (*type II errors*). In other words, if a true

parent-child pair is misclassified, it might indicate that the child has some unexpected facial features that could be caused by a neural developmental disorder.

However, it remains the question if it is even possible to use this approach. The aim of this study is to research the possibility of using kinship verification for this detection task and to identify what its limitations are.

This is done by training a simple classifier for kinship verification first. Then, it is tested on a KdVS dataset. The pictures were preprocessed and transformed into feature vectors using *Local Binary Patterns* (LBP) (He and Wang, 1990) and then classified using a *Support Vector Machine* (SVM) (Cortes and Vapnik, 1995). Additional test sets are constructed to test whether the classifier can generalize and to identify whether it is good enough to transfer its decision model to a different domain.

It should be noted that this method of automatic syndrome detection for diagnosis is aimed to be a tool to aid clinicians in conjunction with existing molecular genetic tests. A lot more factors play a role in diagnosis than our classifier takes into account. Only trained professionals who are aware of the nuances of clinical diagnosis should use it.

2 Related Work

Our approach to syndrome detection lends from previous work in syndrome classification and kinship verification.

2.1 Syndrome classification

This is not the first investigation of the use of artificial intelligence in clinical diagnosis. Zhu, Shang, Shao, and Guo (2017) used a *Deep Convolutional Neural Network* (DCNN) to detect depression in video data. Obafemi-Ajayi et al. (2015) identified clusters of facial features that correlate to the symptoms of *Autistic Spectrum Disorder* (ASD) and built classifiers to test their distinctiveness. Ferry et al. (2014) built a system that could diagnose eight different developmental disorders.

Although these few examples show that automatic diagnosis is feasible, they had access to a substantial amount of data. The novelty in the proposed approach lies in that it does not need any syndrome-specific data to train on.

2.2 Kinship verification

Kinship verification was first introduced by Fang, Tang, Snavely, and Chen (2010). They used different facial and textural features and used a *k-Nearest Neighbors* (kNN) and an SVM as classifier to get a 70.67% and 68.60% accuracy, respectively, on a handmade dataset.

Afterwards, many more algorithms have been proposed that increased performance, aided by the competition for kinship verification in 2014 (Lu, Hu, Zhou, Zhou, Castrillón-Santana, Lorenzo-Navarro, Kou, Shang, Bottino, and Vieira, 2014a). Several benchmark datasets were created to allow better comparison between the different techniques, some of these datasets will be discussed in section 3. Unfortunately, all these new techniques adhere to the specific nature of kinship verification data and are not used in other domains. A selection of them will be named here briefly, but the reader is referred to the original papers for a detailed description.

The approaches to kinship verification can be summarized in three categories; improved feature selection, improved learning algorithms and improved use of data.

Feature selection was improved by (Alirezazadeh, Fathi, and Abdali-Mohammadi, 2018) with an algorithm called *Kinship Feature Selection* (KinFS) that specifically selects facial features relevant for kinship verification. Similarly, Zhou, Hu, Lu, Shang, and Guan (2011) and Zhou, Lu, Hu, and Shang (2012) created *Software Product Line Engineering* (SPLE) and *Gabor-based Gradient Orientation Pyramid* (GGOP) features, respectively. SPLE and GGOP had a human-like accuracy of 67.72% and 69.75% with an SVM on an in-house kinship verification dataset.

The learning algorithms can be grouped into three approaches: metric learning, strategy learning and similarity learning.

Metric learning algorithms focus on learning a measurement that describes the kinship relation. These algorithms include *Neighborhood Repulsed Metric Learning* (NRML), *Multiview NRML* (MNRML) (Lu, Zhou, Tan, Shang, and Zhou, 2014b), *Discriminative Deep Metric Learning* (DDML) and *Discriminative Deep Multi-Metric Learning* (DDMML) (Lu, Hu, and Tan, 2017).

Strategy learning focuses on finding the right strategy of classifying kinship. An example of this is *Online Strategy Learning with Average strategy* (OSL-A) (Xu and Shang, 2016).

Similarity learning is similar to metric learning, but it does not find an explicit metric. Instead, the algorithms learn the metric implicitly. This approach has been used in *Ensemble Similarity Learning* (ESL) (Zhou, Shang, Yan, and Guo, 2016) and *Self-Similarity Representation of Weber faces* (SSRW) (Kohli, Singh, and Vatsa, 2012).

The usage of data in kinship verification can also increase its performance. By using supplementary data, it is possible to leverage the specific dynamics of kinship features.

Xia, Shao, and Fu (2011) leveraged the dynamic that children often have similar facial features as the parents when they were young. The UB KinFace dataset was specifically created with pictures of the parents when they were young and old. This allows the use of transfer learning to account for the age gap between parent and child and improve classification accuracy up to 60.0% (Xia et al., 2011). This is better than the reported human accuracy of 53.7% and 56.0% on the same dataset (Shao, Xia, and Fu, 2011).

Dibeklioglu, Salah, and Gevers (2013) had the insight that family members seem to have similar smiles and smile dynamics. They used videos of smiles for kinship verification. In a later study, they went beyond kinship verification and created a generator that could generate the smile of the potential offspring of someone (Ertugrul and Dibeklioglu, 2017).

Although these methods are not directly used in our approach, which is described in section 4, they do form its basis and they can be applied to it. The latter will be discussed in section 9.

3 Data

Data is not only necessary for training any classifier, it usually constrains the classifier as well. This is because a classifier can only learn patterns that are present in the data it is trained on. Thus, it may learn undesired patterns that are in the dataset and it cannot learn desired patterns that are absent from the dataset. Hence, this section will not only explain why the UB KinFace dataset was chosen and give a description of the Koolen-de Vries (KdVS) dataset that is used for syndrome detection, it will also discuss the effects of the datasets on the expected performance of the classifiers.

Several kinship verification datasets have been developed for benchmarking. However, most of the datasets are created by cropping faces out of family pictures. This introduces an extreme bias into the datasets, as is discussed by López, Boutellaa, and Hadid (2016). These biased datasets are not suitable for transfer learning. Thus our choice of dataset was restricted to only 3 options: the UB KinFace dataset, the Family101 dataset and the Smile DB. The latter only consists of videos and is therefore unsuitable. Since the Family101 dataset is smaller than the UB KinFace dataset, it was chosen to use the UB KinFace dataset.

3.1 UB KinFace dataset

The UB KinFace dataset consists of 200 positive parent-child pairs and is created by searching with Google image search for pictures of public figures (Xia et al., 2011; Xia, Shao, Luo, and Fu, 2012). The parents have two pictures: one when they were young and one when they were older. This means that the dataset contains both gray-scale and color images. The children have one picture each. This creates a dataset with 600 images in total. Each image has a resolution of 127x100 pixels and is annotated with the locations of the eyes, nose and mouth. The dataset was manually extended with gender annotations that can be found in appendix C and on GitHub* for further use. Unfortunately, several biases exist in this dataset. Aside from the selection bias introduced by searching for celebrities on Google, the dataset is skewed towards Asian father-child pairs. Half of the parent-child pairs are of Asian descent and mothers are underrepresented. The latter can be seen in table 3.1a that shows the gender distribution of the UB KinFace dataset.

3.2 UB KinFace split

Usually, a dataset consists of a train and a test set with positive and negative examples. However, the UB KinFace dataset does not contain a predefined train-test split nor any predefined negative

*https://github.com/Aerylia/UBKinFaceGender

Table 3.1: Distribution of genders in the orignal UB KinFace (a) (before creation of negative pairs) and KdVS (b) datasets.

(a) Ub KinFace dataset

UB KinFace	Son	Daughter	Total
Father	93	77	170
Mother	12	18	30
Total	105	95	200

(b) KdVS dataset

KdVS	Son	Daughter	Total
Father	4	4	8
Mother	15	13	28
Total	19	17	36

instances; this had to be created manually. It was also necessary to define the use of the old or young images for the parents, because they could each be used. This situation has been leveraged by defining several datasets that can be used interchangeably. This allowed the classifiers to be tested on their generalizability.

First, the train-test split was defined in an abstract way. The participants (parent-child pairs) in the dataset were randomly split into a train and test set. This is done before the creation of the negative pairs to remove the possibility of information bleed between the sets. The split ratio is 80% train data and 20% test data.

Then, each parent-child pair is assigned a negative parent-child pair, creating the positive-negative pairs. They can be used to define the negative instances for kinship verification, but also allow the creation of negative instances for other tasks (e.g. using both parent images for identification). The positive-negative pairs are created by assigning each parent-child pair another parent-child pair in their respective train or test set using a random permutation. The permutation is adjusted to remove any positive pairs. The positive and negative pairs of the train and test set, respectively, are concatenated and shuffled. The train and test sets can be recreated using the instructions in appendix D. The resulting dataset sizes can be seen in table 3.2.

Using the predefined train-test split and the predefined positive-negative pairs, the UB KinFace dataset is split into 4 different datasets: old-child, young-child, any-child and old-young. The old-child and young-child datasets are created by combining the old, respectively young, parent images and the child images. The any-child dataset is the concatenation of the old-child and young-child datasets. The old-young dataset is created by combining the old parent images with the young parent images. The old-young dataset cannot be used for kinship verification and therefore its train set is not used. Negative instances of these datasets are created by combining the parent images with corresponding negative images from the respective positive-negative pair. The creation process of the different train and test sets is shown schematically in figure 3.1.

Table 3.2: Number of parent-child pairs in the train and test sets for each gender group of the UB KinFace dataset after creation of negative pairs.

Train/Test split	Train	Test	Total
Father - son	145	30	175
Father - daughter	125	36	161
Mother - son	23	8	31
Mother - daughter	27	6	33
Total	320	80	400

3.3 KdVS dataset

For syndrome classification, a dataset of images was collected by the Human Genetics Department of the RadboudUMC. The pictures in the dataset were taken at a conference for parents of children with KdVS. The gender distribution of the parent-child pairs is shown in table 3.1b. All pictures are of caucasians and taken under the same conditions. Due to the nature of the conference, most subjects wore the same t-shirt. All children in the dataset have been diagnosed with KdVS and a paternity test was performed to ensure soundness of the data.

The choice of the UB KinFace dataset for kinship verification and the KdVS dataset for syndrome detection has a significant effect on the results. This will be discussed in section 7. Suggestions for improvement to the construction process of the KdVS dataset are given in section 9.

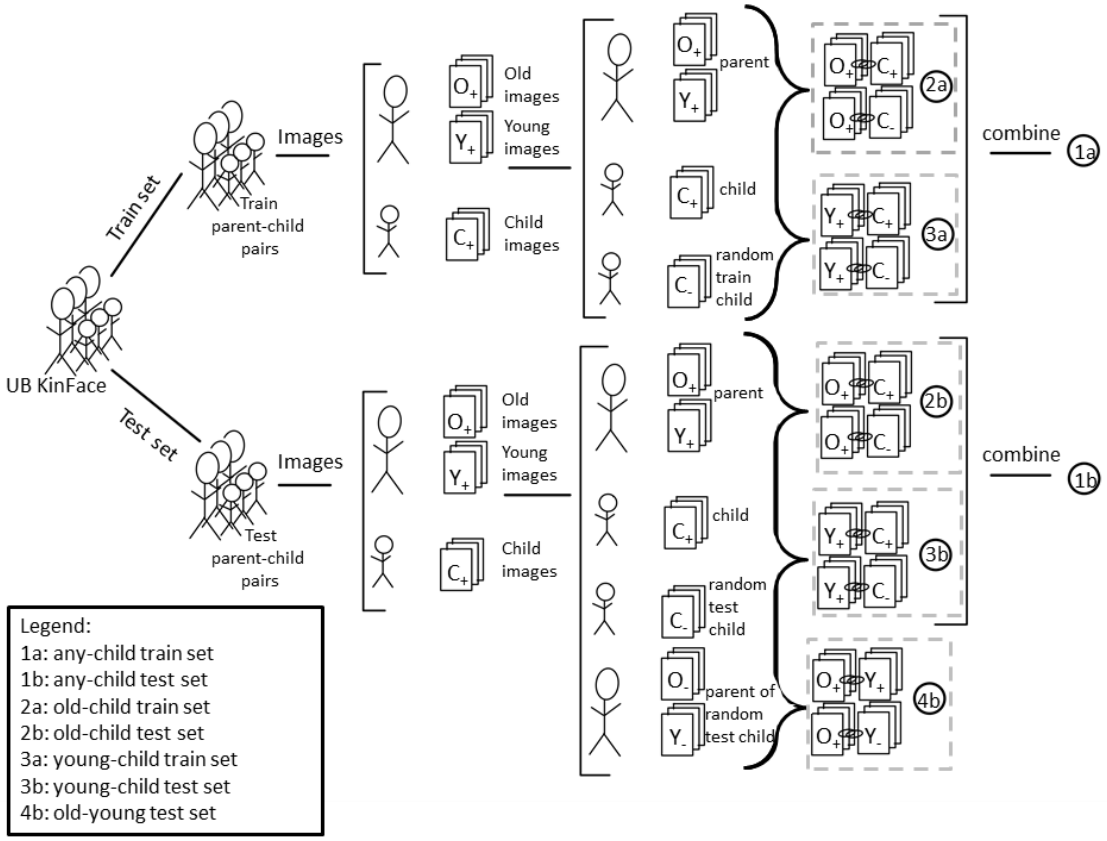

Figure 3.1: A visualization of how the different train and test sets are created from the UB KinFace dataset. The parent-child pairs are split into a train and test set with their corresponding old parent images (O_+), young parent images (Y_+) and child images (C_+). Negative pairs are created by assigning each parent-child pair a fixed random parent-child pair (O_-, Y_-, C_-). The images are paired to create the old-child train ($2a$) and test ($2b$) set, the young-child train ($3a$) and test ($3b$) set, and the any-child train ($1a$) and test ($1b$) set. The positive pairs in the old-young test set ($4b$) are created by pairing the old image of each parent to its young image. The negative pairs are created by coupling the old images with the young image of the assigned random parent.

4 Methods

A key component in this research is that the approach needs to be able to perform well with little training data. Even the existing kinship verification datasets are too small to properly train a neural network. The use of a state-of-the-art algorithm for our kinship verifier would be premature optimization, because it is not clear if the transfer learning approach could work. Instead, a proof-of-concept is created using a simple algorithm that can be extended in future research once its effectiveness is determined.

4.1 Preprocessing

The kinship classifier needs the input images to be comparable. This is done by preprocessing the images first. The faces on the images are made to be equal size by normalizing the images on eye distance and rotating them such that the eyes are aligned horizontally. Then, the images are cropped to have the same size. Lastly, all color images are converted to gray-scale to make the color space uniform across all images.

4.2 Feature selection

The pictures are transformed into a feature vector using a technique called *Local Binary Patterns* (LBP) (He and Wang, 1990). LBP can describe the texture of a face (Ahonen, Hadid, and Pietikainen, 2006) and has been used in other kinship verification algorithms for obtaining features (Fang et al., 2010; Lu et al., 2014b; Zhou et al., 2016; Lu et al., 2017, among others), as well as for a baseline for comparison (Zhou et al., 2011, 2012, among others).

LBP uses the difference of pixel intensity in the (e.g. 3x3) neighborhood of each pixel to encode the local texture in an image into a pattern. These patterns are then grouped per block of n by m pixels to create a histogram of their occurrences. Each histogram is represented in a feature vector. These feature vectors are then concatenated to create a single feature vector (Ahonen et al., 2006).

LBP, in its pure form, will create a very sparse feature vector, because the amount of different patterns is $2^8 = 256$ and the blocks generally do not have that many pixels. To solve this, two extensions to LBP have been proposed: rotation invariant and uniform patterns (Ojala, Pietikainen, and Maenpaa, 2002).

Rotation invariant (ri) patterns are patterns that stay the same regardless of rotation. This means that the original local binary patterns are grouped in the histogram when they would be the same if a different neighborhood pixel was chosen to start with. Rotation invariance reduces the number of histogram bins to 36.

Uniform patterns (u2) are defined to be patterns with at most 2 transitions from 0 to 1 and vice versa. The histogram is created with a bin for each unique uniform pattern and a bin for all non-uniform patterns, that is $58 + 1 = 59$ bins in total.

The two extensions can also be combined (**riu2**) by creating a histogram with bins for each *rotation invariant uniform pattern* and a bin for the remaining patterns. This reduces the histogram to a mere 10 bins.

A parent-child pair is a combination of the feature vectors of the parent image and the child image. The features vectors can be combined by concatenating them or by calculating their difference. Using the difference of feature vectors should improve performance as the classifier would not need to learn the 1:1 correspondence between the features in the vectors. However, if kinship is not described by the difference between the same features, the classifier would perform better when concatenating the feature vectors, as it could find new feature relations.

If the resulting feature vector dimension is larger than 200, it is reduced using *Principal Component Analysis* (PCA) to a 200 dimensional feature vector.

It was empirically chosen to use 8x8 blocks, the rotation invariant LBP patterns and the difference of feature vectors using a 3-fold cross validated grid search. This results in a feature vector with a dimension of $8*8*36 = 2304$, which is reduced to 200 with PCA. The results on which these decisions are based can be found in appendix A.

4.3 Classifier

The obtained feature vectors are classified using a set of *Support Vector Machines* (SVM), which have been shown to perform well on kinship verification (Yan, Lu, Deng, and Zhou, 2014). The SVM uses a Radial Basis Function (RBF) kernel, since the difference of kin versus non-kin relations in the face is most likely non-linear. The 'C' and 'γ' hyper parameters of the SVMs are chosen using a 3-fold cross validated grid search over the sets $[10^i | i \in [-10, \ldots, 3]]$ for C and $[10^i | i \in [-10, \ldots, 3]]$ for γ. The results can be found in appendix A. The parameters were picked based on the highest average performance of the five classifiers.

This approach has similarities with the first kinship verification work (Fang et al., 2010) and the approach of one of the participants of the kinship verification competition (KVW14) (Lu et al., 2014a).

5 Experiments

In order to make any conclusions about the performance of the kinship verification classifiers and their usefulness for syndrome detection with the proposed method, several experiments have been carried out.

All classifiers in the experiments have the structure as described in section 4. They were trained

using the different UB KinFace sub-datasets as detailed in section 3.

Due to the nature of the construction of these sets, they can be used interchangeably for training and testing without the risk of information bleed or selection bias. This allows an investigation of the classifiers that goes beyond the accuracy on a held-out validation set through different experiments.

These experiments can be separated into 3 distinct tasks: *kinship verification*, *identification* and *syndrome detection*. Where each task can be performed with gender information (*gender-specific*) or without gender information (*genderless*).

Separate classifiers were trained on the any-child, old-child and young-child subsets of the UB KinFace dataset, as explained in section 3. These classifiers are called the any-child classifiers, old-child classifiers and young-child classifiers, respectively, for ease.

For each (sub)dataset, 5 SVMs are trained, one SVM for the genderless kinship verification task (*parent-child*) and four SVMs for each gender specific task (*father-son* (f-s), *father-daughter* (f-d), *mother-son* (m-s), *mother-daughter* (m-d)). This separation of gender is made with the idea that fathers and sons look alike in a different way than fathers and daughters, etcetera. To generalize over this difference, a lot of training data is needed.

5.1 Kinship verification task

The goal of this research is to use kinship verification for syndrome detection. To do this, it is necessary that the used kinship verifier has a high accuracy. This measure is obtained by testing each trained classifier on their respective test set.

It is also possible to investigate if the classifiers generalize well to a slightly different dataset due to the method of construction of the different train sets. This means that the old-child classifiers can be tested on the young-child data, and vice versa, to see if they can generalize of age.

Similarly, a sanity-check can be performed by testing the any-child classifiers on the old-child and young-child data. The average performance on the old-child and and young-child data should be equal to the performance on the any-child data, since it is the same data.

5.2 Identification task

Another way of using the UB KinFace dataset is to use the kinship verifier to do facial identification of the parents. A kinship verifier should be able to match facial features of the parent to those of the child. Because these facial features are genetically caused, it should also be able to match these features of a parent to those of a younger version of the parent.

It may be possible that a kinship verifier finds a different pattern to distinguish the parent and child. Although this can still be a good classifier for kinship verification, this classifier would not be suitable for syndrome detection using our transfer learning approach.

The performance of the classifiers on the identification task is investigated by testing them on the old-young dataset.

5.3 Syndrome detection task

Traditionally, transfer learning is done by continued training of the classifier with a new dataset. However, this technique is not suitable for SVMs. Instead, the trained kinship verification classifiers are used directly for the syndrome detection task.

The ability of the classifiers to perform this task is investigated by testing them on the KdVS dataset. To this end, the used annotation for each KdVS parent-child pair is the ground truth: $True$. The goal of this experiment is that the classifiers have a low accuracy.

6 Results

This section discusses the performance of the different classifiers on the different tasks in relation to each other.

6.1 Kinship verification results

The results are shown in table 6.1. At first glance, the classifiers do not have a state-of-the-art accuracy on the kinship verification task. However, most of them do seem to perform slightly better than chance.

All genderless classifiers were unable to distinguish the kinship relation. Although their performance is the same, the any-child classifier (see ta-

Table 6.1: Accuracy of the classifiers trained on the any-child (a), old-child (b) and young-child (c) train sets, respectively.

(a) any-child classifiers

Test set	all	f-s	f-d	m-s	m-d	mean
any-child	0.5000	0.5333	0.5278	0.6250	1.0000	0.6715
old-child	0.5000	0.5333	0.5278	0.6250	1.0000	0.7028
young-child	0.5000	0.5333	0.5278	0.6250	1.0000	0.6403
old-young	0.5000	0.5593	1.0000	0.5714	0.8571	0.7470
KdVS	0.0000	1.0000	0.0000	0.2500	0.6154	0.4664

(b) old-child classifiers

Test set	all	f-s	f-d	m-s	m-d	mean
old-child	0.5000	0.5333	0.5278	0.6250	0.6667	0.5882
young-child	0.5000	0.5333	0.5278	0.6250	0.6667	0.5882
old-young	0.5000	0.5593	1.0000	1.0000	1.0000	0.8898
KdVS	1.0000	1.0000	0.0000	0.0000	1.0000	0.5000

(c) young-child classifiers

test set	all	f-s	f-d	m-s	m-d	mean
young-child	0.5000	0.5333	0.5278	0.6250	0.6667	0.5882
old-child	0.5000	0.5333	0.5278	0.6250	0.6667	0.5882
old-young	0.5000	0.5593	1.0000	1.0000	1.0000	0.8898
KdVS	1.0000	1.0000	0.0000	0.0000	1.0000	0.5000

ble 6.1a) found a different relation than the other classifiers (table 6.1b and 6.1c) since its accuracy on the KdVS data is different.

The average of the any-child classifier on the old-child and young-child test sets is indeed the average of its performance on the any-child test set, as expected.

The average accuracy of the gender-specific classifiers, regardless of training set (column **mean** in the tables of table 6.1), is better than the accuracy of the respective genderless classifiers on kinship verification. This confirms that the assumption that the facial features that indicate kinship are different depending on gender. Similarly, the gender-specific classifiers perform better if the child has the same gender as the parent (f-s, m-d) compared to having the opposite gender (f-d and m-s, respectively).

Noticeably, most gender-specific classifiers perform the same on the test sets, respectively, as can be seen in the tables of table 6.1. This may be caused by the small size of their respective train and test sets. However, he confusion matrices of all classifiers, in appendix B, show that all classifiers learned to map all image pairs to either $True$ or $False$, except for the m-s and m-d any-child classifiers. This is peculiar since mothers are underrepresented in the data, thus these classifiers are trained on very little data. A possible explanation of these results will be discussed in section 7.1.

The generalizability of the classifiers on kinship verification can also be seen in tables 6.1b and 6.1c. The accuracy of the old-child classifiers on the young-child test set does not drop in accuracy and vice versa. This would have implied that the classifiers are able to generalize over age, if the classifiers had learned a pattern.

6.2 Identification results

On the identification task, the lower accuracy shows that none of the genderless classifiers generalize well over age. Since this task is very similar to the kinship verification task, it may imply that the genderless classifiers perform at chance level. This is visible from the results on the old-young test set in each table in table 6.1.

Since the confusion matrices in appendix B show that most classifiers map all images pairs to only one of the two classes, only the results of the any-child m-s and m-d classifiers is discussed.

Although the performance of the classifiers on the identification task drops with respect to the kinship verification task, the accuracy is still above chance level. Their confusion matrices can also be found in appendix B, in table B.6. This shows that the classifiers that were able to do kinship verification, were also able to generalize over age. Note that this may be due to the presence of both the young and old parent images in the training data.

6.3 Syndrome detection results

On the KdVS data, the two any-child m-d and m-s classifiers have a worse accuracy on the KdVS dataset than the different kinship verification sets. As can be seen in the results on the KdVS dataset for each classifier (table 6.1a). This is desired, because it is desired that the classifiers have a lower accuracy. The performance of the m-s classifier is even better than on the kinship data ($1 - 0.25 = 0.75$). This does not hold for the m-d classifier ($1 - 0.6154 = 0.3846$). The confusion matrices can be found in appendix B as well, in table B.7.

This may imply that the genderless classifiers were able to learn facial features that are relevant for kinship verification, but those features are less

present in parent-child pairs where the child has KdVS. However, it could also be the case that this missing feature from the KdVS data represents something unrelated to kinship, like the background. The latter option is very plausible because the better kinship verifier (m-d classifier) performed worse on the KdVS data.

7 Discussion

It can be observed from the results that there is room for improvement in our approach. This section will discuss the effects of some of our decisions in five components of our method: the UB KinFace dataset, the KdVS dataset, the method of feature selection, the viability of transfer learning and the limitations of our approach.

7.1 UB KinFace dataset

The performance of our classifiers depends heavily on the quality of their training data. Unfortunately, the results seem to indicate some problem in the UB KinFace dataset. Only the any-child m-d and m-s classifiers were able to learn the kinship relation, even though they were trained on the smallest any-child datasets. This could imply that father-child kinship verification is more difficult than mother-child verification, possibly because a child's face has more female features. Similarly, the genderless kinship verification task is difficult as well due to the lack of gender information. However, it could also be caused by false-positives in the dataset. The parent-child pairs were added to the dataset because it is common knowledge that they are parent and child, but this was not genetically confirmed. It is possible that some of the "fathers" are in fact not the biological father of the children. Assuming that the gender effect on children and parents is the same, it is reasonable to expect that the father-son and father-daughter classifiers perform similar to the mother-daughter and mother-son classifiers, respectively, but this is not visible in the results.

Another possible problem with the data labeling is due to the absence of negative parent-child pairs. Negative pairs are needed for training the classifiers and had to be constructed randomly. Since the dataset is not annotated with the identity of the parents and children, it cannot be checked if some parent-child pairs are related to other pairs. If such relationships exist in the UB KinFace dataset, it is possible that the dataset contains false negative instances. This would make it more difficult to train a classifier while already having so little data.

7.2 KdVS dataset

Similarly, the KdVS dataset constrains the conclusions that can be drawn about the performance of the classifiers for two reasons. First, the images are taken in a highly controlled environment. And second, there is no control group of images of children without KdVS that were taken under the same conditions. This combination of uniform images and the lack of a control group does not allow conclusions to be made about whether the results are caused by KdVS or the controlled environment. Although it is understandable that the clinicians that built this dataset wanted to control as many variables as possible, this situation was avoidable.

When creating a new dataset, it should be discussed what the implications are of the acquisition-method on the nature of the data and their usability in machine learning applications. It is possible that the way in which the dataset is obtained introduces a bias (see López et al., 2016) or limits the usability of that data, as in this case. However, as datasets are generally created for some purpose, it should be stressed how important it is to consider these effects in advance.

In section 9.1, a different acquisition method will be discussed as well as another approach to KdVS detection that can be done using this new dataset.

7.3 Feature extraction

In addition to the possible problems with the used datasets, the way in which the images were treated could also have implications on their performance.

The images were normalized on the eye distance. However, eye distance is not the same for everyone. Since one of the facial features of KdVS is that the eyes are further apart (Koolen et al., 2016), such a normalization method introduces a bias in the KdVS data. This causes all other facial features to be closer to each other in the normalized KdVS data and thus makes the kinship verification task harder. As this difference in facial features was the starting point of the approach, this effect is desired,

but it should be taken into consideration as it might have ripple effects.

An example of such a ripple effect can be found in the decision to take the difference of the feature representations of the two images of a parent-child pair. The values in the feature vectors correspond to the same area in the image, but this may not be the same area in the face. Although it was empirically found that the classifiers performed better on the difference than the concatenation of the feature vectors (see appendix A), this performance difference was very small. So it could be the case that the difference of the feature vectors is not representative for the difference between the facial features.

Then again, for this to be the case, the feature vectors themselves need to be representative for the facial features in the first place and LBP may not be the best descriptor for that. LBP was chosen because it can describe the texture of the face very well (Ahonen et al., 2006), but it is very sensitive to noise, rotation and lighting (Nava, Cristóbal, and Escalante-Ramírez, 2012), which can be problematic when training on images taken under uncontrolled conditions. Another shortcoming of LBP is that the aggregation of the patterns in histograms loses some of the spacial information that may be important for kinship verification (Zhou et al., 2011).

7.4 Kinship verification for syndrome detection

It is imperative for this application that the kinship verification classifiers have high accuracy and sensitivity. Otherwise, the patterns learned by the classifiers are not generalizable enough to be transferred into a new domain. However, kinship verification is a very difficult problem. The reported human performance on the UB KinFace dataset is 53.7% and 56.0% (Shao et al., 2011), which is only slightly better than chance. When taking in consideration that the UB KinFace dataset consists of public figures, it is very possible that some of the participants actually recognized some of the public figures and knew who their parent or child was. That would mean that the actual human performance on kinship verification may be even lower, i.e. at chance level.

Kinship verification is not only difficult for humans, but also for computers. It is an undersampled problem (Yan et al., 2014), the datasets are too small and the variation in facial images is caused by more than genetics alone (Lu et al., 2014b). At best, simple kinship verification algorithms perform just above chance on the UB KinFace dataset (Shao et al., 2011). A lot needs to improve in the domain of kinship verification to be able to use it for transfer learning, especially for our intended application.

However, even if a good kinship verification algorithm existed, it may not be suitable for syndrome detection. The performance of the classifiers on the KdVS data has a strong dependence on how representative the UB KinFace dataset is of the KdVS data. The KdVS data contains only Caucasian subjects and most of the parents are mothers, whereas less than half of the UB KinFace dataset is Caucasian and most of the parents are fathers, as can be seen in table 3.1 (Shao et al., 2011). This could cause that the classifiers are less good at kinship verification on Caucasians than Asians or better on fathers than mothers.

Another difference with the KdVS data is that the UB KinFace dataset consists of pictures of public figures (e.g. celebrities and politicians, see Shao et al., 2011). Aside from the possibility that these people may not have a representative appearance, the images themselves may not be representative either. They may have been adjusted, e.g. with tools like Photoshop, or the subjects took more care to look good on the pictures. This could introduce another bias into the classifiers.

7.5 Unsuitable for diagnosis

In the end, the results show that most kinship verification classifiers are not accurate enough for diagnostic purposes. All genderless, father-son and father-daughter classifiers, as well as the old-child and young-child classifiers, were not able to learn any kinship relation.

The any-child mother-daughter and mother-son classifiers were able to learn a pattern. The mother-daughter classifier was even able to distinguish the two classes with perfect accuracy. However, their results on the KdVS data is inconclusive. Since there is no control group, it is not possible to identify whether the classifiers misclassifications are due to KdVS itself or due to some other factor that causes them.

The latter is very likely because the better classifier makes less type II errors, indicating that the low accuracy of the mother-son classifier on the KdVS data is caused by a poor classifier. This is substantiated by the confusion matrix in table B.4a that shows that most misclassifications of the any-child mother-son classifier are false-negatives; the classification mistakes that are desired for syndrome detection.

However, the biggest shortcoming of our approach, by far, is that it will never be able to detect KdVS on its own. It might be able to indicate if the child phenotype is different from what is expected given the phenotype of the parent, but this is the case for many syndromes (e.g. Down Syndrome). Thus this approach to syndrome detection gives an ambiguous indication.

8 Conclusion

The goal of this research was to identify if it is possible to use a kinship verification classifier for diagnostic purposes.

The UB KinFace dataset was used to train classifiers for kinship verification and tested on three tasks: kinship verification, facial identification and syndrome detection.

The classifiers were trained on two versions of the kinship verification task: genderless and gender-specific. Only the any-child mother-son and mother-daughter classifiers were able to learn the kinship relation with an accuracy of 62.50% and 100.00%, respectively. The conclusions are based on the results from only these two classifiers.

The performance of the two classifiers on the identification task shows that the classifiers may be able to generalize the kinship relation over age, as they perform almost the same on the old-young test set.

However, it is curious that the two classifiers, trained on the smallest dataset, were the only classifiers that learned a pattern in the data. This could be evidence of a problem in the UB KinFace dataset. It is possible that the UB KinFace dataset contains father-child pairs where the father is not genetically related to the child, since this was assumed and not genetically confirmed. If this is the case, then the classifiers trained with father-child pairs would have more difficulty learning the kinship relation, because the two classes cannot be separated. That would result in the observed difference between the any-child classifiers.

The results of the classifiers on the syndrome detection task are inconclusive. This is due to the nature of the KdVS dataset. The images were taken in a highly controlled environment and are not representative of the UB KinFace pictures, which has an effect on the performance of the classifiers on the KdVS dataset. The strength of this effect could have been seen in a control group of pictures, but that data was not available.

Unfortunately, even if the results on the KdVS data were conclusive, the classifier would still not be able to perform the syndrome detection task. The used approach, by definition, is only able to identify deviation from the expected differences of facial features of parents and children. This deviation can be caused by numerous factors, thus it cannot be used for diagnostic purposes.

9 Future work

Although our results may not seem promising at first glance, three approaches are proposed that could be used in a future study.

9.1 Creation of new KdVS dataset

First of all, a new KdVS dataset needs to be created. The conditions of images should not be controlled. However, the pictures do need to adhere to some constraints. To avoid the bias that is present in the KinFaceW I/II datasets (López et al., 2016), the parent and child should not be in the same picture.

This dataset can be created by asking the parents of children with KdVS for personal images (e.g. selfies) of four people: the two (genetically confirmed) parents, the child with KdVS and a sibling who does not have KdVS (if possible).

It is possible that parents impose their own restrictions on this task, which could introduce another bias. This new bias could be identified by adding an image from a fifth and sixth person: an unrelated healthy child and an unrelated healthy adult. This unrelated child could be a friend or neighbor and the adult could be their parent. These

unrelated images can then be used as negative kinship pairs.

Because facial features are strongly influenced by the age of the individual (Xia et al., 2011), it might be an improvement to ask for images of the parents and unrelated adult when they were children.

All pictures should preferably be taken on different dates using different devices and on different locations, as far as this is possible.

This acquisition method should be easier to execute than the original method as older pictures can be used and sent from home.

If these 5 pictures are obtained for each of the 28 children in the original KdVS dataset, then the new dataset would have $28 + 28 = 56$ kin + KdVS pairs (parents and child with KdVS), $28 + 28 = 56$ kin + nonKdVS pairs (parents and sibling), $28 + 28 = 56$ nonkin + nonKdVS pairs (parents and unrelated child) and 28 nonkin + KdVS pairs (unrelated adult and child with KdVS). This dataset would be almost as large as the UB KinFace dataset itself. Although only half of this new dataset can be used for kinship verification, the dataset doubles in size if the nonkin parent and nonkin child are parent and child themselves. It would double in size again when adding the young parent images aswell. With this new dataset, our research could be repeated and improved with better algorithms.

9.2 Statistical diagnosis using deep neural networks

Another, more generalizable, approach could also be tried. The starting point of our research was that individuals with KdVS have different facial features than those without KdVS. It might be possible to identify this difference using a combination of a deep neural network (DNN) and statistical methods.

Deep convolutional neural networks have been used for facial identification with a 97% accuracy (Taigman, Yang, Ranzato, and Wolf, 2014; Sun, Wang, and Tang, 2014). These neural networks are able to identify the key facial features in images that make a person unique. This DNN-knowledge can also be leveraged using transfer learning. This is done by using one (or more) of the hidden layers in the trained DNN as feature vector for another classifier. This approach has been used for facial identification (Sun et al., 2014) and achieved a similar performance as the method without transfer learning (Taigman et al., 2014). Hence, the hidden layers in a trained DNN contain the key facial features and it might be possible to use them for other tasks.

The proposed new approach is to use a hidden layer of a trained deep convolutional neural network, that has state-of-the-art performance on facial identification benchmarks, like Labeled Faces in the Wild (LFW), as feature vector of an image. This representation can be created for pictures of healthy individuals and pictures of those with KdVS. Then, it should be possible to calculate if an image of someone is more likely to belong to the distribution of images of individuals with or without KdVS using statistical methods. It is also possible to use a (statistical) classifier to distinguish the two cases.

This new approach should be usable for syndrome diagnosis and has two additional benefits. One benefit of this approach is that it should be possible to identify which facial features are specific to which class (syndrome). This could help with identifying the typical phenotype of a syndrome. The other benefit is that it is easily extensible to diagnose other syndromes; only a set of examples is needed to indicate its distribution of facial features.

This new approach will work better with more data, but when using pure statistical methods, a small set of examples may already be enough to give an indication.

9.3 Other applications

Both (an adjusted version of) the proposed approach in this paper and the neural feature vector approach could also be used for detecting other syndromes because about 30-40% of all genetic disorders are visible in the face (Ferry et al., 2014). This does not only include disorders that are commonly known to be visible, like Down Syndrome, but also disorders that are more subtly visible, like Autism Spectrum Disorder (Aldridge et al., 2011).

Although these first results do not seem very promising, much can be learned from this research for future studies and applications. It is one more step towards a time where doctors and clinicians need less time for diagnosis, giving them more time to treat their patients.

References

Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.

Kristina Aldridge, Ian D George, Kimberly K Cole, Jordan R Austin, T Nicole Takahashi, Ye Duan, and Judith H Miles. Facial phenotypes in subgroups of prepubertal boys with autism spectrum disorders are correlated with clinical phenotypes. *Molecular Autism*, 2(1):15, 2011.

Pendar Alirezazadeh, Abdolhossein Fathi, and Fardin Abdali-Mohammadi. Effect of purposeful feature extraction in high-dimensional kinship verification problem. *Journal of Computing and Security*, 3(3), 2018.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Hamdi Dibeklioglu, Albert Ali Salah, and Theo Gevers. Like father, like son: Facial expression dynamics for kinship verification. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1497–1504. IEEE, 2013.

Itir Önal Ertugrul and Hamdi Dibeklioglu. What will your future child look like? Modeling and synthesis of hereditary patterns of facial dynamics. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on*, pages 33–40. IEEE, 2017.

Ruogu Fang, Kevin D Tang, Noah Snavely, and Tsuhan Chen. Towards computational models of kinship verification. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 1577–1580. IEEE, 2010.

Quentin Ferry, Julia Steinberg, Caleb Webber, David R FitzPatrick, Chris P Ponting, Andrew Zisserman, and Christoffer Nellåker. Diagnostically relevant facial gestalt information from ordinary photos. *Elife*, 3, 2014.

Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, 28(4): 509–512, 1990.

Naman Kohli, Richa Singh, and Mayank Vatsa. Self-similarity representation of Weber faces for kinship classification. In *Biometrics: Theory, Applications and Systems (BTAS), 2012 IEEE Fifth International Conference on*, pages 245–250. IEEE, 2012.

David A Koolen, Rolph Pfundt, Katrin Linda, Gea Beunders, Hermine E Veenstra-Knol, Jessie H Conta, Ana Maria Fortuna, Gabriele Gillessen-Kaesbach, Sarah Dugan, Sara Halbach, et al. The Koolen-de Vries syndrome: a phenotypic comparison of patients with a 17q21. 31 microdeletion versus a KANSL1 sequence variant. *European Journal of Human Genetics*, 24(5):652, 2016.

Miguel Bordallo López, Elhocine Boutellaa, and Abdenour Hadid. Comments on the kinship face in the wild data sets. *IEEE transactions on pattern analysis and machine intelligence*, 38(11): 2342–2344, 2016.

Jiwen Lu, Junlin Hu, Xiuzhuang Zhou, Jie Zhou, Modesto Castrillón-Santana, Javier Lorenzo-Navarro, Lu Kou, Yuanyuan Shang, Andrea Bottino, and Tiago Figuieiredo Vieira. Kinship verification in the wild: The first kinship verification competition. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–6. IEEE, 2014a.

Jiwen Lu, Xiuzhuang Zhou, Yap-Pen Tan, Yuanyuan Shang, and Jie Zhou. Neighborhood repulsed metric learning for kinship verification. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):331–345, 2014b.

Jiwen Lu, Junlin Hu, and Yap-Peng Tan. Discriminative deep metric learning for face and kinship verification. *IEEE Transactions on Image Processing*, 26(9):4269–4282, 2017.

Rodrigo Nava, Gabriel Cristóbal, and Boris Escalante-Ramírez. A comprehensive study of texture analysis based on local binary patterns. In *Optics, Photonics, and Digital Technologies for Multimedia Applications II*, volume 8436, page 84360E. International Society for Optics and Photonics, 2012.

Tayo Obafemi-Ajayi, Judith H Miles, T Nicole Takahashi, Wenchuan Qi, Kristina Aldridge,

Minqi Zhang, Shi-Qing Xin, Ying He, and Ye Duan. Facial structure analysis separates autism spectrum disorders into meaningful clinical subgroups. *Journal of autism and developmental disorders*, 45(5):1302–1317, 2015.

Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

Ming Shao, Siyu Xia, and Yun Fu. Genealogical face recognition based on UB kinface database. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 60–65. IEEE, 2011.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898, 2014.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

Siyu Xia, Ming Shao, and Yun Fu. Kinship verification through transfer learning. In *IJCAI Proceedings-international joint conference on artificial intelligence*, volume 22, page 2539, 2011.

Siyu Xia, Ming Shao, Jiebo Luo, and Yun Fu. Understanding kin relationships in a photo. *IEEE Transactions on Multimedia*, 14(4):1046–1056, 2012.

Min Xu and Yuanyuan Shang. Kinship verification using facial images by robust similarity learning. *Mathematical Problems in Engineering*, 2016, 2016.

Haibin Yan, Jiwen Lu, Weihong Deng, and Xiuzhuang Zhou. Discriminative multimetric learning for kinship verification. *IEEE Transactions on Information forensics and security*, 9(7): 1169–1178, 2014.

Xiuzhuang Zhou, Junlin Hu, Jiwen Lu, Yuanyuan Shang, and Yong Guan. Kinship verification from facial images under uncontrolled conditions. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 953–956. ACM, 2011.

Xiuzhuang Zhou, Jiwen Lu, Junlin Hu, and Yuanyuan Shang. Gabor-based gradient orientation pyramid for kinship verification under uncontrolled environments. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 725–728. ACM, 2012.

Xiuzhuang Zhou, Yuanyuan Shang, Haibin Yan, and Guodong Guo. Ensemble similarity learning for kinship verification from facial images in the wild. *Information Fusion*, 32:40–48, 2016.

Yu Zhu, Yuanyuan Shang, Zhuhong Shao, and Guodong Guo. Automated depression diagnosis based on deep networks to encode facial appearance and dynamics. *IEEE Transactions on Affective Computing*, 2017.

A Results parameter search

This appendix contains the gridsearch results for the kinship verification classifiers on the UB KinFace dataset. The best results are given for each LBP mapping and block size. The results contain a validation accuracy (Val score) as well as a cross-validated accurcay (CV score). The column "best params" shows the C parameter and kernel for the SVM that would give the best accuracy according to the gridsearch. For every experiment, the kernel was set to 'rbf'.

The first section shows the results when the feature vectors of parent and child are concatenated and the second section uses the difference of these features.

A.1 Results concatenation

Table A.1: Gridsearch results for genderless kinship verification on KinFace (Concatenated).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5	0.5	"'C': 1e-10, 'gamma': 0.1, 'kernel': 'rbf'"
riu2	4x5	0.5	0.5	"'C': 1e-10, 'gamma': 0.1, 'kernel': 'rbf'"
riu2	7x5	0.5	0.5078125	"'C': 1e-10, 'gamma': 0.01, 'kernel': 'rbf'"
riu2	8x8	0.5	0.5015625	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
riu2	10x10	0.5	0.509375	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
ri	1x1	0.5	0.5	"'C': 1e-10, 'gamma': 0.1, 'kernel': 'rbf'"
ri	4x5	0.5	0.503125	"'C': 1e-07, 'gamma': 0.01, 'kernel': 'rbf'"
ri	7x5	0.5	0.509375	"'C': 1e-10, 'gamma': 0.01, 'kernel': 'rbf'"
ri	8x8	0.49375	0.5046875	"'C': 1, 'gamma': 0.001, 'kernel': 'rbf'"
ri	10x10	0.5	0.509375	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
u2	1x1	0.5125	0.50625	"'C': 1e-05, 'gamma': 0.001, 'kernel': 'rbf'"
u2	4x5	0.48125	0.525	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	7x5	0.5	0.5265625	"'C': 1e-10, 'gamma': 0.01, 'kernel': 'rbf'"
u2	8x8	0.5	0.521875	"'C': 1e-10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	10x10	0.5	0.5140625	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
all	all	0.5	0.5	"'C': 1e-10, 'gamma': 0.001, 'kernel': 'rbf'"

Table A.2: Gridsearch results for kinship verification on the father-son pairs of KinFace (Concatenated).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5166666666666667	0.5172413793103449	"'C': 100, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	4x5	0.55	0.5172413793103449	"'C': 10, 'gamma': 1e-08, 'kernel': 'rbf'"
riu2	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	8x8	0.5333333333333333	0.5137931034482759	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
riu2	10x10	0.5333333333333333	0.5172413793103449	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
ri	1x1	0.5166666666666667	0.5172413793103449	"'C': 1, 'gamma': 1e-08, 'kernel': 'rbf'"
ri	4x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	8x8	0.5333333333333333	0.5241379310344828	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
ri	10x10	0.5333333333333333	0.5241379310344828	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	1x1	0.5333333333333333	0.5172413793103449	"'C': 1, 'gamma': 1e-08, 'kernel': 'rbf'"
u2	4x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	8x8	0.5333333333333333	0.5448275862068965	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	10x10	0.5333333333333333	0.5586206896551724	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
all	all	0.5166666666666667	0.5206896551724138	"'C': 10, 'gamma': 1e-09, 'kernel': 'rbf'"

Table A.3: Gridsearch results for kinship verification on the father-daughter pairs of KinFace (Concatenated).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	4x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	8x8	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	10x10	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	1x1	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	4x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	8x8	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	10x10	0.5277777777777778	0.52	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	1x1	0.5416666666666666	0.516	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	4x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	8x8	0.5277777777777778	0.528	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	10x10	0.5277777777777778	0.536	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
all	all	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"

Table A.4: Gridsearch results for kinship verification on the mother-son pairs of KinFace (Concatenated).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.75	0.6956521739130435	"'C': 1000, 'gamma': 1e-09, 'kernel': 'rbf'"
riu2	4x5	0.6875	0.782608695652174	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	7x5	0.6875	0.7391304347826086	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	8x8	0.625	0.8478260869565217	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	10x10	0.6875	0.8043478260869565	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	1x1	0.6875	0.6956521739130435	"'C': 1000, 'gamma': 1e-08, 'kernel': 'rbf'"
ri	4x5	0.6875	0.7608695652173914	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	7x5	0.6875	0.7391304347826086	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	8x8	0.625	0.8478260869565217	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	10x10	0.75	0.8043478260869565	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	1x1	0.625	0.6521739130434783	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	4x5	0.625	0.7391304347826086	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	7x5	0.5	0.7391304347826086	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	8x8	0.625	0.7608695652173914	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	10x10	0.5625	0.7608695652173914	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
all	all	0.6875	0.6956521739130435	"'C': 100, 'gamma': 1e-08, 'kernel': 'rbf'"

Table A.5: Gridsearch results for kinship verification on the mother-daughter pairs of KinFace (Concatenated).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.75	0.6111111111111112	"'C': 10, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	4x5	0.6666666666666666	0.6111111111111112	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
riu2	7x5	0.6666666666666666	0.5740740740740741	"'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'"
riu2	8x8	1.0	0.6111111111111112	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
riu2	10x10	0.75	0.6111111111111112	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	1x1	0.75	0.5925925925925926	"'C': 10, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	4x5	0.6666666666666666	0.5925925925925926	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
ri	7x5	0.6666666666666666	0.5925925925925926	"'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'"
ri	8x8	1.0	0.6111111111111112	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	10x10	0.6666666666666666	0.6111111111111112	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
u2	1x1	0.6666666666666666	0.6296296296296297	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	4x5	0.6666666666666666	0.6296296296296297	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	7x5	0.8333333333333334	0.6666666666666666	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	8x8	1.0	0.6666666666666666	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	10x10	0.6666666666666666	0.6481481481481481	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
all	all	0.6666666666666666	0.5925925925925926	"'C': 1, 'gamma': 1e-07, 'kernel': 'rbf'"

A.2 Results difference

Table A.6: Gridsearch results for genderless kinship verification on KinFace (Difference).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5	0.5015625	"'C': 1e-10, 'gamma': 0.0001, 'kernel': 'rbf'"
riu2	4x5	0.5	0.5	"'C': 1e-10, 'gamma': 0.1, 'kernel': 'rbf'"
riu2	7x5	0.5	0.5125	"'C': 1, 'gamma': 0.01, 'kernel': 'rbf'"
riu2	8x8	0.5	0.503125	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
riu2	10x10	0.5	0.5015625	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
ri	1x1	0.5	0.503125	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
ri	4x5	0.5	0.5	"'C': 1e-10, 'gamma': 0.1, 'kernel': 'rbf'"
ri	7x5	0.5	0.5078125	"'C': 1e-10, 'gamma': 0.01, 'kernel': 'rbf'"
ri	8x8	0.5	0.503125	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
ri	10x10	0.4875	0.5046875	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
u2	1x1	0.49375	0.503125	"'C': 1e-10, 'gamma': 0.001, 'kernel': 'rbf'"
u2	4x5	0.51875	0.53125	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	7x5	0.5	0.525	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	8x8	0.5	0.5203125	"'C': 1e-10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	10x10	0.5	0.525	"'C': 10, 'gamma': 0.001, 'kernel': 'rbf'"
all	all	0.5	0.5	"'C': 1e-10, 'gamma': 0.001, 'kernel': 'rbf'"

Table A.7: Gridsearch results for kinship verification on the father-son pairs of KinFace (Difference).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	4x5	0.5833333333333334	0.5241379310344828	"'C': 1, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	8x8	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	10x10	0.5333333333333333	0.5137931034482759	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
ri	1x1	0.55	0.5137931034482759	"'C': 10, 'gamma': 1e-08, 'kernel': 'rbf'"
ri	4x5	0.6	0.5310344827586206	"'C': 10, 'gamma': 1e-08, 'kernel': 'rbf'"
ri	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	8x8	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	10x10	0.5333333333333333	0.5275862068965518	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	1x1	0.55	0.5241379310344828	"'C': 10, 'gamma': 1e-09, 'kernel': 'rbf'"
u2	4x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	7x5	0.5333333333333333	0.5103448275862069	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	8x8	0.5333333333333333	0.5241379310344828	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
u2	10x10	0.5333333333333333	0.5482758620689655	"'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'"
all	all	0.5666666666666667	0.5275862068965518	"'C': 10, 'gamma': 1e-10, 'kernel': 'rbf'"

Table A.8: Gridsearch results for kinship verification on the father-daughter pairs of KinFace (Difference).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.5416666666666666	0.516	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	4x5	0.5277777777777778	0.516	"'C': 1, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	8x8	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	10x10	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	1x1	0.5	0.532	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	4x5	0.5277777777777778	0.516	"'C': 1, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	8x8	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	10x10	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	1x1	0.4583333333333333	0.536	"'C': 1, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	4x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	7x5	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	8x8	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	10x10	0.5277777777777778	0.528	"'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'"
all	all	0.5277777777777778	0.512	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"

Table A.9: Gridsearch results for kinship verification on the mother-son pairs of KinFace (Difference).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.4375	0.6956521739130435	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	4x5	0.5625	0.6956521739130435	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	7x5	0.625	0.6521739130434783	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
riu2	8x8	0.5625	0.7608695652173914	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
riu2	10x10	0.5625	0.6739130434782609	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	1x1	0.5625	0.6521739130434783	"'C': 100, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	4x5	0.6875	0.6739130434782609	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	7x5	0.625	0.6521739130434783	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	8x8	0.625	0.717391304347826	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
ri	10x10	0.625	0.6521739130434783	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	1x1	0.5625	0.6086956521739131	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	4x5	0.625	0.6956521739130435	"'C': 10, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	7x5	0.625	0.6521739130434783	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
u2	8x8	0.625	0.6956521739130435	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
u2	10x10	0.6875	0.717391304347826	"'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'"
all	all	0.5	0.6956521739130435	"'C': 10, 'gamma': 1e-07, 'kernel': 'rbf'"

Table A.10: Gridsearch results for kinship verification on the mother-daughter pairs of KinFace (Difference).

Mapping	Block	Val score	CV score	Best params
riu2	1x1	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	4x5	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	7x5	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
riu2	8x8	1.0	0.6111111111111112	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
riu2	10x10	0.75	0.6296296296296297	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	1x1	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	4x5	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	7x5	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
ri	8x8	1.0	0.6296296296296297	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
ri	10x10	0.8333333333333334	0.6296296296296297	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	1x1	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"
u2	4x5	0.75	0.6296296296296297	"'C': 1000, 'gamma': 1e-08, 'kernel': 'rbf'"
u2	7x5	0.6666666666666666	0.6666666666666666	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
u2	8x8	0.6666666666666666	0.6296296296296297	"'C': 1000, 'gamma': 1e-07, 'kernel': 'rbf'"
u2	10x10	0.75	0.6851851851851852	"'C': 100, 'gamma': 1e-06, 'kernel': 'rbf'"
all	all	0.6666666666666666	0.5555555555555556	"'C': 1e-10, 'gamma': 1e-10, 'kernel': 'rbf'"

B Confusion matrices

This appendix contains the confusion matrices for all classifiers whose performance is discussed in section 6. The matrices are separated into kinship verification results, identification results and syndrome detection results.

B.1 Kinship verification task

Table B.1: Confusion matrices of the genderless classifiers on their respective kinship verification test set.

(a) any-child genderless classifier

		Classification	
		True	**False**
Ground	**Kin**	0	80
truth	¬ **Kin**	0	80

(b) old-child genderless classifier

		Classification	
		True	**False**
Ground	**Kin**	40	0
truth	¬ **Kin**	40	0

(c) young-child genderless classifier

		Classification	
		True	**False**
Ground	**Kin**	40	0
truth	¬ **Kin**	40	0

Table B.2: Confusion matrices of the f-s classifiers on their respective kinship verification test set.

(a) any-child f-s classifier

		Classification	
		True	**False**
Ground	**Kin**	32	0
truth	¬ **Kin**	28	0

(b) old-child f-s classifier

		Classification	
		True	**False**
Ground	**Kin**	16	0
truth	¬ **Kin**	14	0

(c) young-child f-s classifier

		Classification	
		True	**False**
Ground	**Kin**	16	0
truth	¬ **Kin**	14	0

Table B.3: Confusion matrices of the f-d classifiers on their respective kinship verification test set.

(a) any-child f-d classifier

		Classification	
		True	**False**
Ground	**Kin**	0	34
truth	¬ **Kin**	0	38

(b) old-child f-d classifier

		Classification	
		True	**False**
Ground	**Kin**	0	17
truth	¬ **Kin**	0	19

(c) young-child f-d classifier

		Classification	
		True	**False**
Ground	**Kin**	0	17
truth	¬ **Kin**	0	19

Table B.4: Confusion matrices of the m-s classifiers on their respective kinship verification test set.

(a) any-child m-s classifier

		Classification	
		True	**False**
Ground	**Kin**	2	4
truth	¬ **Kin**	2	8

(b) old-child m-s classifier

		Classification	
		True	**False**
Ground	**Kin**	0	5
truth	¬ **Kin**	0	3

(c) young-child m-s classifier

		Classification	
		True	**False**
Ground	**Kin**	0	5
truth	¬ **Kin**	0	3

Table B.5: Confusion matrices of the m-d classifiers on their respective kinship verification test set.

(a) any-child m-d classifier

		Classification	
		True	**False**
Ground	**Kin**	8	0
truth	¬ **Kin**	0	4

(b) old-child m-d classifier

		Classification	
		True	**False**
Ground	**Kin**	4	0
truth	¬ **Kin**	2	0

(c) young-child m-d classifier

		Classification	
		True	**False**
Ground	**Kin**	4	0
truth	¬ **Kin**	2	0

B.2 Identification task

Table B.6: Confusion matrices of the m-d and m-s any-child classifiers on the identification task (old-young test set).

(a) identification any-child m-s classifier

		Classification	
		True	**False**
Ground	**Kin**	1	2
truth	¬ **Kin**	2	3

(b) identification any-child m-d classifier

		Classification	
		True	**False**
Ground	**Kin**	6	1
truth	¬ **Kin**	0	0

B.3 Syndrome detection task

Table B.7: Confusion matrices of the m-d and m-s any-child classifiers on the syndrome detection task (KdVS test set).

(a) m-s classifier

		Classification	
		True	**False**
Ground	**Kin**	1	3
truth	¬ **Kin**	0	0

(b) m-d classifier

		Classification	
		True	**False**
Ground	**Kin**	8	5
truth	¬ **Kin**	0	0

C KinFace gender annotation

This appendix contains the manually annotated gender of the UB KinFace dataset that was used for this research. It can also be found on GitHub: https://github.com/Aerylia/UBKinFaceGender.

Table C.1: Gender annotation for the parent and child in KinFace.

Id (resp. file)	Parent gender	Child gender
1.jpg	m	m
2.jpg	f	m
3.jpg	m	m
4.jpg	m	f
5.jpg	f	f
6.jpg	m	f
7.jpg	f	m
8.jpg	m	m
9.jpg	m	m
10.jpg	m	m
11.jpg	m	m
12.jpg	m	m
13.jpg	m	m
14.jpg	m	f
15.jpg	m	m
16.jpg	m	f
17.jpg	m	m
18.jpg	m	m
19.jpg	m	m
20.jpg	m	m
21.jpg	m	m
22.jpg	m	f
23.jpg	m	m
24.jpg	m	f
25.jpg	m	f
26.jpg	m	m
27.jpg	f	f
28.jpg	m	m
29.jpg	m	m
30.jpg	m	f
31.jpg	m	m
32.jpg	m	f
33.jpg	m	m
34.jpg	m	m
35.jpg	m	m
36.jpg	f	f
37.jpg	m	f
38.jpg	m	m
39.jpg	f	f
40.jpg	m	f
41.jpg	m	m

. . . continued

Id (resp. file)	Parent gender	Child gender
42.jpg	f	f
43.jpg	m	m
44.jpg	m	f
45.jpg	f	f
46.jpg	m	m
47.jpg	m	f
48.jpg	f	m
49.jpg	m	m
50.jpg	m	f
51.jpg	m	m
52.jpg	m	f
53.jpg	m	f
54.jpg	m	f
55.jpg	m	m
56.jpg	f	m
57.jpg	f	m
58.jpg	f	f
59.jpg	f	f
60.jpg	m	f
61.jpg	f	f
62.jpg	m	m
63.jpg	m	m
64.jpg	m	m
65.jpg	f	m
66.jpg	m	f
67.jpg	m	m
68.jpg	m	f
69.jpg	m	m
70.jpg	m	f
71.jpg	m	m
72.jpg	m	f
73.jpg	m	m
74.jpg	m	f
75.jpg	m	f
76.jpg	m	m
77.jpg	m	m
78.jpg	m	f
79.jpg	m	m
80.jpg	m	m
81.jpg	m	m
82.jpg	m	m
83.jpg	m	m
84.jpg	m	f
85.jpg	m	m
86.jpg	f	m
87.jpg	f	f

. . . continued

Id (resp. file)	Parent gender	Child gender
88.jpg	f	m
89.jpg	f	f
90.jpg	m	m
91.jpg	m	m
92.jpg	m	m
93.jpg	m	m
94.jpg	m	f
95.jpg	m	f
96.jpg	m	m
97.jpg	m	f
98.jpg	m	f
99.jpg	m	m
100.jpg	m	m
101.jpg	f	f
102.jpg	m	f
103.jpg	m	f
104.jpg	m	f
105.jpg	m	m
106.jpg	f	m
107.jpg	m	m
108.jpg	m	m
109.jpg	m	m
110.jpg	m	m
111.jpg	m	f
112.jpg	m	m
113.jpg	m	f
114.jpg	m	f
115.jpg	f	f
116.jpg	m	f
117.jpg	f	f
118.jpg	m	m
119.jpg	m	f
120.jpg	f	m
121.jpg	m	f
122.jpg	m	m
123.jpg	m	m
124.jpg	m	m
125.jpg	m	f
126.jpg	m	m
127.jpg	m	m
128.jpg	m	f
129.jpg	m	f
130.jpg	m	m
131.jpg	m	f
132.jpg	m	m
133.jpg	m	m

. . . continued

Id (resp. file)	Parent gender	Child gender
134.jpg	m	m
135.jpg	m	m
136.jpg	m	m
137.jpg	m	m
138.jpg	m	m
139.jpg	m	f
140.jpg	m	f
141.jpg	m	f
142.jpg	m	f
143.jpg	f	f
144.jpg	m	m
145.jpg	m	m
146.jpg	f	m
147.jpg	m	f
148.jpg	m	f
149.jpg	m	m
150.jpg	f	f
151.jpg	m	f
152.jpg	m	f
153.jpg	m	f
154.jpg	m	f
155.jpg	m	m
156.jpg	m	m
157.jpg	m	m
158.jpg	m	m
159.jpg	m	m
160.jpg	m	m
161.jpg	m	f
162.jpg	m	f
163.jpg	m	f
164.jpg	m	f
165.jpg	m	f
166.jpg	m	m
167.jpg	m	m
168.jpg	m	f
169.jpg	m	f
170.jpg	m	f
171.jpg	m	f
172.jpg	m	f
173.jpg	m	f
174.jpg	m	m
175.jpg	m	m
176.jpg	m	f
177.jpg	m	f
178.jpg	m	f
179.jpg	m	f

. . . continued

Id (resp. file)	Parent gender	Child gender
180.jpg	m	m
181.jpg	m	f
182.jpg	m	f
183.jpg	m	f
184.jpg	m	m
185.jpg	m	f
186.jpg	m	f
187.jpg	f	f
188.jpg	m	f
189.jpg	f	f
190.jpg	m	f
191.jpg	f	m
192.jpg	m	f
193.jpg	m	f
194.jpg	m	m
195.jpg	m	m
196.jpg	m	f
197.jpg	m	m
198.jpg	m	m
199.jpg	f	f
200.jpg	f	m

D Train - test split description

This appendix contains the python code for creating the different train and test sets that can be combined to create the old-child, young-child, any-child and old-young datasets. The shuffling is given explicitly to allow for perfect replication. Note that shuffling the test set is not strictly necessary.

Python code for creating the sub datasets of UB KinFace.

```
# Definition of the indexes
train_idxs = [139, 90, 182, 58, 5, 94, 37, 22, 193, 51, 21, 161, 63, 34, 1, 97, 87, 75,
    91, 176, 164, 54, 33, 29, 138, 15, 166, 135, 189, 102, 123, 36, 198, 50, 13, 55, 76,
     120, 184, 152, 134, 47, 88, 65, 199, 89, 20, 163, 30, 84, 0, 78, 148, 155, 130,
    180, 185, 9, 160, 142, 126, 25, 175, 68, 107, 95, 100, 143, 72, 122, 48, 42, 157,
    181, 79, 156, 62, 32, 177, 67, 172, 2, 147, 11, 41, 8, 117, 44, 17, 115, 118, 12,
    45, 10, 127, 61, 171, 151, 70, 188, 112, 24, 141, 121, 38, 144, 186, 96, 4, 39, 92,
    85, 149, 131, 53, 192, 28, 103, 168, 59, 108, 27, 93, 165, 174, 194, 64, 16, 190,
    125, 60, 43, 52, 145, 71, 137, 98, 14, 19, 195, 187, 114, 3, 49, 128, 86, 18, 111,
    169, 154, 173, 56, 162, 69, 106, 73, 99, 167, 133, 196]
test_idxs = [153, 46, 132, 170, 119, 101, 6, 23, 77, 183, 197, 104, 83, 35, 191, 140, 7,
     179, 40, 124, 159, 150, 81, 113, 116, 82, 80, 158, 110, 57, 26, 66, 109, 146, 105,
    136, 129, 178, 74, 31]

neg_train_children = [25, 145, 64, 120, 164, 37, 115, 28, 32, 180, 12, 53, 162, 1, 187,
    172, 20, 73, 59, 114, 84, 111, 55, 89, 2, 118, 17, 138, 156, 54, 141, 39, 22, 95,
    49, 135, 24, 60, 195, 196, 38, 92, 94, 137, 51, 127, 198, 97, 76, 13, 173, 199, 142,
     123, 56, 190, 98, 152, 177, 193, 42, 58, 93, 103, 63, 181, 143, 33, 69, 14, 188,
    79, 70, 131, 27, 0, 108, 100, 9, 154, 10, 16, 52, 184, 130, 41, 44, 144, 87, 175,
    155, 106, 168, 102, 8, 186, 99, 176, 107, 72, 194, 3, 90, 117, 85, 182, 19, 165, 86,
     30, 133, 29, 78, 139, 5, 43, 122, 148, 65, 126, 147, 128, 71, 47, 75, 88, 18, 68,
    161, 149, 67, 11, 125, 21, 169, 171, 91, 157, 62, 185, 50, 192, 15, 36, 48, 134,
    174, 189, 34, 61, 4, 160, 96, 151, 163, 167, 45, 121, 112, 166]
neg_test_children = [178, 35, 31, 105, 159, 132, 191, 136, 83, 109, 46, 7, 23, 129, 183,
     66, 101, 74, 81, 104, 179, 153, 57, 124, 80, 197, 26, 116, 146, 140, 82, 110, 113,
    77, 40, 6, 170, 158, 119, 150]

train_shuffle = [323, 172, 627, 534, 254, 432, 566, 203, 603, 593, 101, 107, 179, 509,
    512, 344, 244, 246, 383, 303, 563, 29, 562, 397, 266, 446, 19, 571, 294, 316, 567,
    569, 497, 173, 293, 309, 598, 180, 450, 614, 158, 589, 18, 427, 104, 594, 171, 123,
    343, 336, 81, 322, 635, 62, 47, 393, 39, 333, 149, 31, 514, 296, 226, 302, 193, 11,
    241, 306, 170, 255, 455, 638, 615, 115, 235, 466, 40, 165, 92, 69, 133, 319, 291,
    227, 408, 281, 353, 378, 586, 198, 508, 116, 349, 422, 375, 464, 518, 592, 240, 125,
     150, 341, 576, 539, 232, 304, 236, 108, 145, 328, 470, 345, 377, 248, 229, 192,
    477, 570, 498, 416, 476, 284, 267, 68, 431, 606, 212, 357, 190, 494, 493, 275, 280,
    515, 204, 220, 436, 315, 478, 30, 79, 505, 368, 490, 468, 417, 146, 106, 22, 412,
    544, 607, 24, 35, 262, 467, 67, 358, 348, 633, 418, 103, 156, 399, 365, 484, 75, 84,
     411, 580, 120, 196, 385, 433, 588, 391, 482, 465, 113, 536, 53, 245, 234, 549, 16,
    325, 230, 622, 251, 491, 76, 122, 137, 530, 58, 109, 277, 276, 168, 318, 605, 428,
    23, 507, 177, 425, 324, 600, 272, 506, 577, 337, 216, 147, 33, 290, 46, 454, 552,
    548, 613, 388, 575, 487, 522, 222, 335, 199, 525, 619, 134, 414, 483, 28, 72, 618,
    312, 537, 415, 210, 243, 152, 400, 545, 80, 578, 38, 200, 205, 488, 264, 342, 438,
    326, 298, 111, 144, 558, 409, 233, 381, 148, 191, 579, 286, 228, 444, 535, 217, 503,
     102, 495, 71, 407, 632, 153, 117, 584, 384, 295, 214, 352, 159, 20, 27, 623, 480,
    460, 59, 631, 66, 195, 499, 531, 329, 60, 340, 213, 64, 360, 4, 164, 462, 118, 132,
    361, 140, 129, 51, 299, 183, 85, 347, 26, 207, 176, 597, 612, 305, 582, 583, 481,
    372, 591, 403, 442, 287, 32, 206, 98, 346, 97, 610, 215, 288, 390, 386, 105, 114,
    338, 225, 625, 223, 380, 1, 307, 160, 599, 268, 10, 501, 486, 239, 492, 41, 83, 285,
     471, 630, 231, 45, 617, 188, 99, 13, 572, 387, 88, 561, 211, 70, 556, 313, 9, 63,
    44, 249, 413, 174, 269, 138, 565, 472, 405, 189, 73, 517, 93, 331, 520, 419, 394,
    36, 396, 317, 43, 437, 392, 382, 628, 185, 550, 475, 554, 447, 0, 379, 456, 91, 181,
     42, 389, 52, 135, 474, 143, 524, 540, 263, 221, 169, 274, 453, 161, 17, 136, 258,
    110, 511, 332, 259, 452, 89, 90, 194, 224, 373, 354, 401, 15, 187, 602, 516, 424,
    369, 8, 590, 626, 513, 54, 175, 541, 426, 163, 608, 3, 457, 459, 166, 479, 208, 321,
```

```
219, 327, 124, 574, 555, 55, 197, 261, 112, 157, 529, 355, 242, 37, 131, 596, 178,
441, 130, 500, 637, 141, 448, 458, 485, 573, 320, 445, 609, 201, 252, 250, 34, 95,
151, 351, 449, 519, 402, 253, 311, 126, 142, 87, 289, 620, 616, 528, 366, 439, 5,
595, 551, 463, 543, 636, 370, 601, 435, 398, 49, 527, 553, 489, 350, 271, 339, 533,
301, 356, 167, 237, 292, 260, 430, 611, 25, 367, 238, 639, 202, 61, 404, 538, 121,
182, 279, 634, 74, 420, 371, 314, 423, 629, 406, 461, 128, 443, 624, 568, 48, 127,
7, 56, 2, 154, 300, 559, 184, 585, 77, 6, 395, 57, 265, 496, 96, 560, 421, 604, 621,
273, 473, 186, 162, 374, 504, 362, 542, 330, 410, 434, 218, 440, 364, 510, 502, 86,
119, 139, 359, 65, 523, 521, 247, 376, 564, 155, 283, 581, 547, 12, 256, 557, 270,
278, 308, 429, 526, 100, 310, 78, 94, 469, 82, 334, 546, 363, 587, 14, 21, 209, 532,
50, 451, 282, 257, 297]
test_shuffle = [80, 56, 152, 138, 40, 68, 155, 66, 13, 130, 47, 60, 129, 73, 5, 114,
139, 18, 79, 65, 19, 110, 143, 48, 57, 49, 104, 137, 46, 127, 43, 107, 118, 51, 28,
109, 136, 94, 74, 131, 39, 151, 132, 121, 72, 62, 24, 37, 25, 6, 58, 52, 78, 16,
122, 145, 153, 150, 113, 141, 142, 35, 61, 84, 105, 36, 108, 96, 64, 148, 30, 3, 10,
42, 44, 32, 0, 85, 146, 100, 125, 135, 140, 154, 147, 97, 7, 67, 21, 38, 31, 27,
82, 99, 81, 93, 86, 14, 22, 9, 126, 89, 101, 11, 144, 88, 119, 117, 83, 158, 149,
33, 115, 54, 156, 70, 45, 134, 69, 34, 92, 77, 8, 71, 128, 12, 4, 53, 90, 26, 29,
106, 17, 133, 120, 41, 55, 91, 1, 20, 124, 50, 75, 116, 157, 2, 87, 102, 98, 23,
111, 59, 95, 15, 103, 123, 76, 159, 112, 63]
# Get the preprocessed features for the old parent, young parent and child pictures.
# The list index is the id of the parent/child.
old_features = get_old_features()
young_features = get_young_features()
child_features = get_child_features()

# Define the old, young and all train dataset.
old_train_set = combine_pair(old_features[train_idxs], child_features[train_idxs]) +
    combine_pair(old_features[train_idxs], child_features[neg_train_children])
young_train_set = combine_pair(young_features[train_idxs], child_features[train_idxs]) +
     combine_pair(young_features[train_idxs], child_features[neg_train_children])
all_train_set = old_train_set + young_train_set

# Define the old, young and all test dataset.
old_test_set = combine_pair(old_features[test_idxs], child_features[test_idxs]) +
    combine_pair(old_features[test_idxs], child_features[neg_test_children])
young_test_set = combine_pair(young_features[test_idxs], child_features[test_idxs]) +
    combine_pair(young_features[test_idxs], child_features[neg_test_children])
all_test_set = old_test_set + young_test_set

# define the old-young test dataset
old_young_test_set = combine_pair(old_features[test_idxs], young_features[test_idxs]) +
    combine_pair(old_features[test_idxs], young_features[neg_test_children])

# Shuffle the different train and test sets.
old_train_set = old_train_set[[i for i in train_shuffle if i < len(old_train_set)]]
young_train_set = young_train_set[[i-len(young_train_set) for i in train_shuffle if i >=
     len(young_train_set)]]
all_train_set = all_train_set[train_shuffle]

old_test_set = old_test_set[[i for i in test_shuffle if i < len(old_test_set)]]
young_test_set = young_test_set[[i-len(young_test_set) for i in test_shuffle if i >= len
    (young_test_set)]]
all_test_set = all_test_set[test_shuffle]
old_young_test_set = old_young_test_set[[i for i in test_shuffle if i < len(old_test_set
    )]]

# Define the sub datasets: (train, test)
old_child = old_train_set, old_test_set
young_child = young_train_set, young_test_set
any_child = all_train_set, any_test_set
old_young = None, old_young_test_set
```