ŠIAULIAI UNIVERSITY
INSTITUTE OF REGIONAL DEVELOPMENT

Lukas Vaitkevičius

Detection of Scribbles Elements by Image Recognition

MASTERS THESIS

Supervisor: doc. dr. D. Dervinis
Reviewer: doc. dr. N. Ramanauskas

Šiauliai 2019

„Tvirtinu, jog darbe pateikta medžiaga nėra plagijuota ir paruošta naudojant literatūros sąraše pateiktus informacinius šaltinius bei savo tyrimų duomenis“

Darbo autoriaus:	Lukas Vaitkevičius
	(vardas, pavardė, parašas)

„I declare that all material is my own work, not plagiarised, and fully acknowledged in the reference list if adapted from other sources. “

Author	Lukas Vaitkevičius
	(name, surname, signature)

SANTRAUKA

Autorius: Lukas Vaitkevičius

Tema: Keverzonių elementų atpažinimas vaizdų apdorojimo metodu

Šiaulių universitetas 2019

Darbo tikslas – surasti arba sukurti keverzonių atpažinimui tinkantį metodą. Buvo išbandyti keturi mašininio mokymo metodai:

- K artimiausių kaimynų metodas su dinaminiu laiko skalės iškreipimo atstumu
- Atraminių vektorių mašinos
- Atsitiktinis miškas
- Konvoliucinis neuroninis tinklas

Artimiausių kaimynų metodas pasirodė netinkamas dėl keverzonių sekų ir R. Kellogg sukurtų klasių ypatybių. Klasės buvo sukurtos pagal vaikų piešimo ir asmenybės raidą.

Atraminių vektorių mašinų ir atsitiktinio miško metodai pasiekia neblogą tikslumą, nepaisant to, kad įvesties duomenys galbūt galėtų būti geriau suglaudinti. Atsitiktinio miško metodas vis dėlto reikšmingai lenkia atraminių vektorių mašinų metodą, greičiu ir tikslumu.

Konvoliucinių neuroninių tinklų metodas pasirodė geriausiai. Duomenų augmentacijos dėka, iš 7037 keverzonių sukuriamos papildomos 148,029 keverzonės. Taip apėjus konvoliucinio tinklo permokymo problemą, metodas pasiekia apie 98% tikslumą.

Raktiniai žodžiai: vaikų keverzonės, piešimo analizė, mašininis mokymas, klasifikacija, dinaminis laiko skalės iškreipimas, k-artimiausi kaimynai, atraminių vektorių mašinos, atsitiktinis miškas, konvoliuciniai neuroniniai tinklai.

SUMMARY

Author: Lukas Vaitkevičius

Subject: Detection of Scribbles Elements by Image Recognition

Šiauliai University 2019

The goal is to find or create a method suitable for scribble recognition. Four overall machine learning methods were tested:

- K Nearest Neighbour using DTW distance measure
- Support Vector Machines
- Random Forest
- Convolutional Neural Networks

KNN DTW is proven to be unsuitable due to the nature of vectorised drawing data and the arbitrary class distinctions imposed by R. Kellogg, who naturally approached the problem of classification from a children's psychological development perspective.

Support vector machine and Random Forest methods perform admirably despite minimal feature engineering, with Random Forest proving superior, reaching competitive accuracy and classification speed.

Finally, convolutional neural networks are shown to handle scribbles as time-series-like data well, except for significant overfitting, which was solved by taking advantage of data augmentation. Most of R. Kellogg's classes are easy to augment, bringing the dataset from 7037 scribble instances to 148,029. The best accuracy of this classifier is approximately 98%.

Keywords: children scribbles, drawing analysis, machine learning, classification, dynamic time warping, k-nearest neighbours, support vector machines, random forest, convolutional neural networks.

AIM AND TASKS

Aim - To create and analyse potential methods for scribble recognition and select the best candidate.

Tasks:

- Analyse methods suitable for scribble classification
- Acquire and prepare scribble data
- Adapt and apply classification methods to the task of scribble recognition
- Test methods to find the best candidate

Supervisor: doc. dr. D. Dervinis
(name, surname, signature)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

There has been interest in children drawings since the end of the 19th century. Researchers (ex. Kellogg[18], Widlocher[34], Arnheim[2], Lowenfeld[22]) have given much attention to the analysis of children drawings. According to them, such analysis can tell much about the mental development of children, based on the drawer's age and the features, details of the work. However, there is lack of consensus between researchers, as to what drawing features should be attributed to what age of development. For example, Widlocher[34] claims, that the first scribbles can be made by children 18 months old, meanwhile Lowenfield[22] states that they're a feature of drawings by children 2 – 3 years of age. According to Widlocher[34], children begin with diagonal lines, but to R. Kellogg, they first master horizontal and vertical lines, and only then move on to other angles.

These inconsistencies make it difficult to use children drawings as a measure of their development, thus more research into correlation between age and hand-eye coordination is required. To this end, it could be useful for researchers to have a tool for drawing analysis; in this particular case, scribble classification. Currently, there is no software readily available to aid in drawing analysis.

There has been much research in the area of human drawing through digital input interfaces. Starting with Sutherland and his Sketchpad system[31], that used a light pen to allow graphical input. Since then, digitizer pen tablets have become widely available and affordable computer input tools.

Most works can be placed in one of two categories:

Recognition with raster image input, such as using convolutional networks for sketch classification[35], statistical models[29], or hand-drawn symbol recognition[9]

Vector graphics, or pen stroke input recognition, for example, scribble recognizer as input interface[7,8], scribble gesture recognition as input[20], recurrent neural network used in categorized scribble generation[13].

Recognition of scribble categories as defined by R. Kellogg is a similar problem, but these specific classes have not been used in automatic recognition, in particular, using vectorized single-stroke input. This work sets out to find a viable approach by testing several distinct methods in order to a predictor with most favourable properties.

2. THEORY

2.1. Classes of scribbles

R. Kellogg, has analysed a little under a million of children drawings[18]. She determined that children between the age of 2 and 3 develop their drawing ability by scribbling in specific patterns. She has come up with a distinct set of scribble classes, which will be used as labels for the scribble classifier. Her classification system is in Figure 2.1.

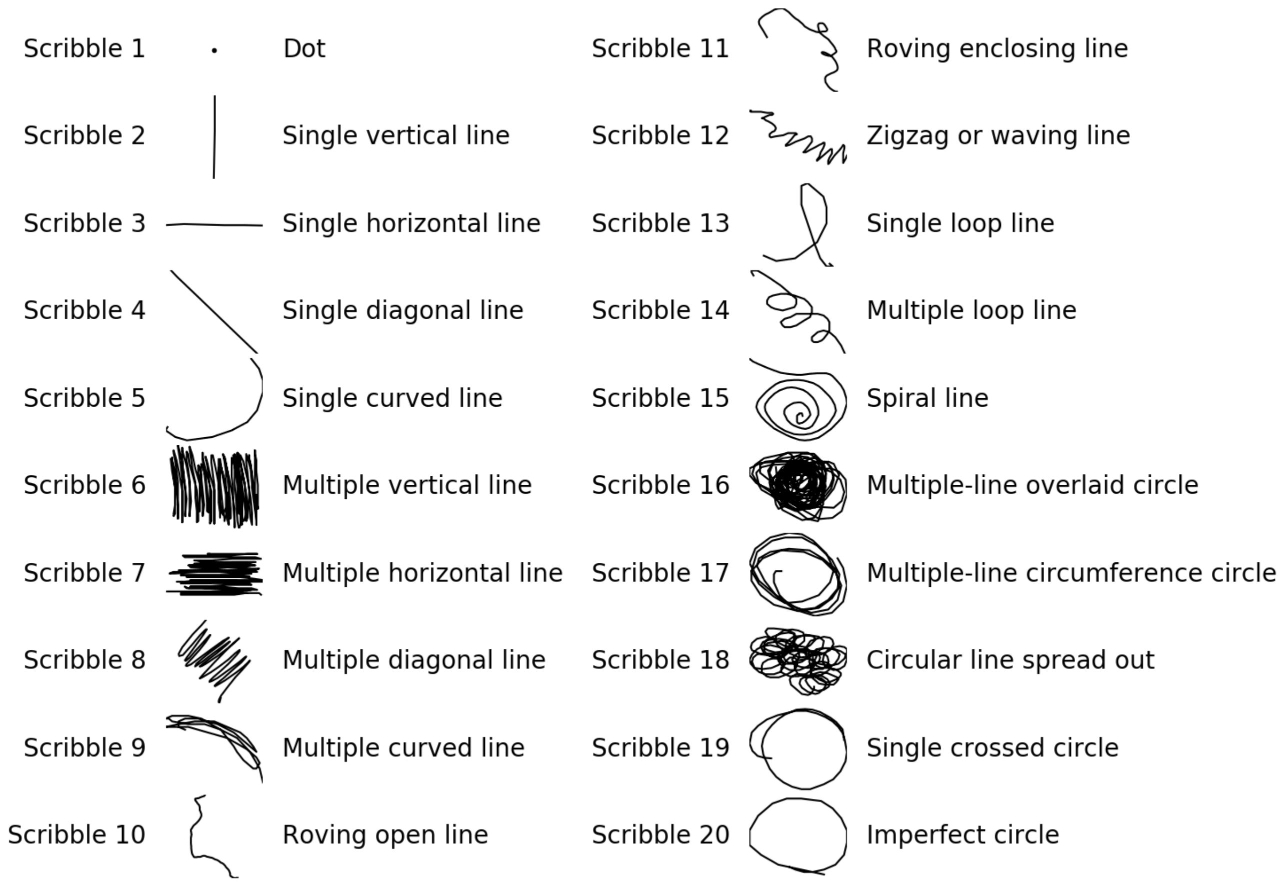


Fig. 2.1. Rhoda Kellogg`s scribble classification system.

2.2. Drawing data

Drawings for analysis are acquired via Drawing Recorder software[11]. The program captures user actions, such that the entire drawing process can be recreated. This has several advantages and disadvantages.

A special set up is required for drawing acquisition, namely a digitizer tablet connected to a computer running the recorder software, better if the tablet can double as an interactive screen. Furthermore, this precludes the use of scanned or photographed drawings for analysis, which eliminates a lot of potential samples for data.

On the other hand, recording the drawing process digitally allows for extracting more information from a single drawing session. Without the need for digitizing physical drawings, there is no noise from scanning or photographing the drawings. Recording the drawing process itself adds a time dimensionality to the data, as opposed to a still 2D image that could be acquired by digitizing a physical drawing. Having a time dimension avoids the loss of information within a drawing, in case elements are drawn on top of one another, and enables the handling of drawing strokes as time-series data.

2.2.1. Drawing data acquisition

Drawing data is categorized by events:

- Drawing/pen movement
- Brush size change
- Brush colour change

In the programming context, mouse events are used, but in this case the mouse can be any pointing device, whether a physical mouse or a digitizer tablet stylus.

Drawing/movement data is captured on left button down event, and until the button up event fires, signifying the end of the stroke either by mouse or pen. On each tick of program execution, the values for pointer coordinates, pressure (when using stylus, otherwise 0) and milliseconds from last execution tick are saved within a list. One such list represents a drawing stroke. Recording the time between each point of capture is useful, because the software is likely to run at varying frequencies between different machines, so this variation can be accounted for if necessary. In practice, for various reasons, these intervals are irregular even on the same machine.

This series of data is stored as a list inside the drawing event, alongside a timestamp for the mouse down/up event for macro time tracking.

Modifications of the brush state are saved as separate events, such that the current colour and stroke width of a line is determined by how the brush parameters were set last.

R. Kellogg's classified scribbles can be digitized by redrawing them with the Drawing Recorder software. Each stroke can be saved as a separate set of data to be used as a template for later comparison. Examples of drawn and recreated scribbles in Figure 2.2.

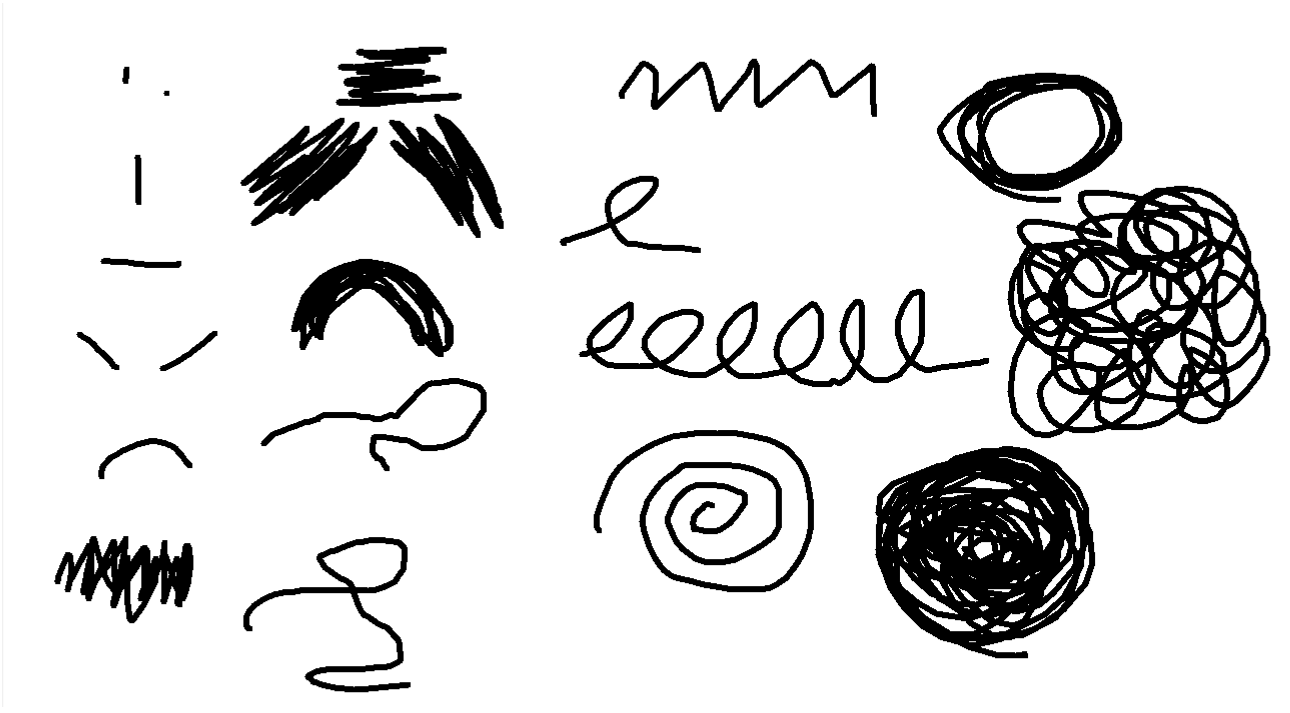


Fig. 2.2. Samples of Rhoda Kellogg`s classes created in Drawing Recorder software.

2.2.2. Scribble data specifics

A drawing, when created within the drawing process recorder, is essentially a collection of two-dimensional time-series representing brush or pen strokes. For the purposes of scribble analysis, an assumption is made that generally a single stroke will equate to a single scribble. In these strokes, the 2D coordinate information is most significant, with brush size having some significance for the resulting shape, and the colour component being rather irrelevant to a scribble unless multiple adjoining strokes are considered as potentially a single scribble.

Due to the nature of drawing and the way it is captured within software, certain features are expected to vary between samples of the same scribble class.

Start position. Each stroke has a starting coordinate, while this might commonly be the top left corner of the resulting stroke shape, in theory it can be within any part of a scribble.

Stroke location. Captured coordinates are relative to the digital canvas in the recorder software, as each stroke can be anywhere within the canvas, coordinates between two strokes can, as a collection, can be arbitrary distances apart.

Scribble angle. Scribbles can be drawn at an arbitrary angle. While R. Kellogg's classifies some scribbles by their shape and angle (classes 2,3,4,6,7,8, Figure 2.3), other classes are angle invariant (Figure 2.4).

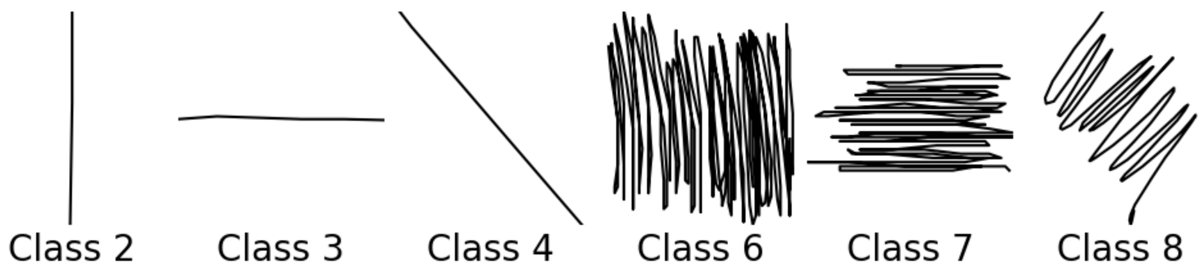


Fig. 2.3. Angle/rotation dependent scribble classes

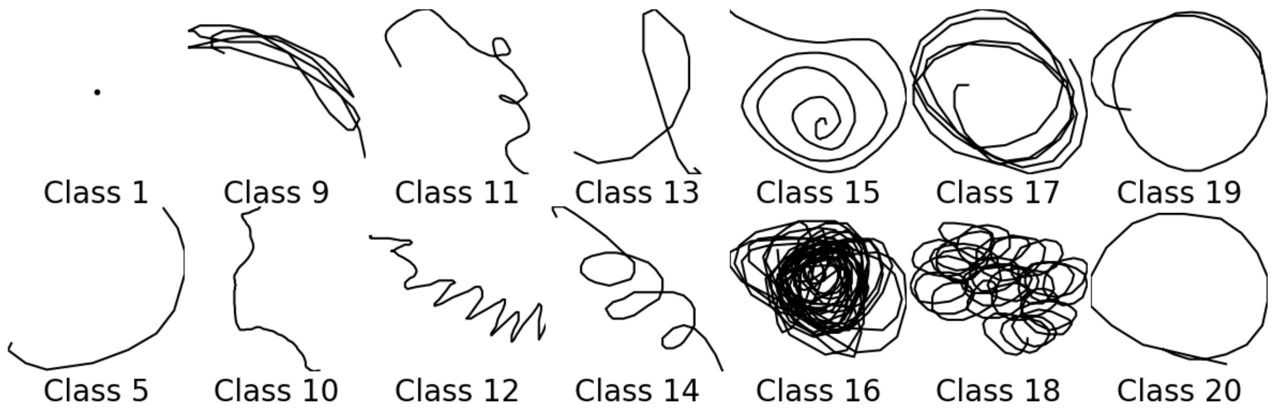


Fig. 2.4. Angle/rotation invariant scribble classes.

2.3. Comparison algorithms

2.3.1. Dynamic Time Warping

DTW is an algorithm for measuring distance (similarity) between two time-series. It is commonly used for evaluating differences between two time-series signals, because of its ability to compensate for shift and distortion in the time domain[23]. Translating this to the context of a drawing stroke, the DTW metric should be more sensitive to the shape and change in direction of the stroke, rather than length or duration of it.

The Dynamic Time Warping distance has two major steps in its calculation.

First, a matrix *D* is built, *n* by *m*, where *n* is length of the first series *A* and *m* is length of the second series *B*. This matrix is filled according to formula:

$$D_{i,j} = \left|A_i - B_j\right| + \min\left(D_{i-1,j-1}, D_{i,j-1}, D_{i-1,j}\right)$$

Eq. 2.1. Dynamic time warping distance matrix calculation.

Where if *i-1* or *j-1* is not a valid index (such as at the starting edges of the matrix), then the value is considered to be 0.

Next, the algorithm searches for a path from the value $D_{n,m}$, to value $D_{0,0}$ by choosing lowest neighbouring values from three of possible previous matrix elements ($D_{n-1,m-1}$, $D_{n-1,m}$, $D_{n,m-1}$)

until it reaches $D_{0,0}$. Example path in Figure 2.5. After the path is complete, the sum of each element within this path is computed and returned as the DTW's distance metric.

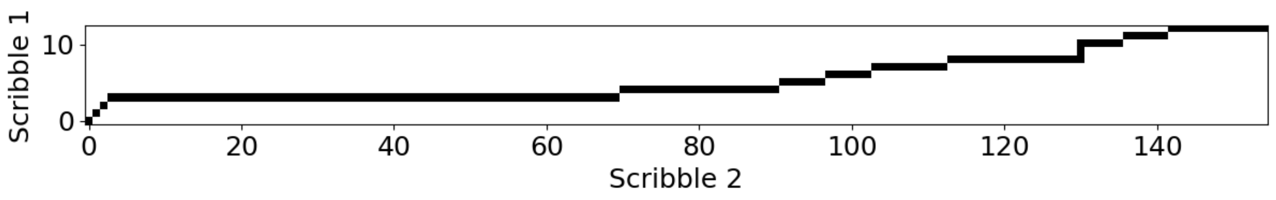


Fig. 2.5. Example of warping path between two scribbles.

2.3.2. Multi-dimensional Dynamic Time Warping

DTW can be generalized to accept multi-dimensional time series in two ways[3014].

Independent Multidimensional DTW computes distance separately for each dimension in subject time-series. This distance is summed to produce the final result.

Dependent Multidimensional DTW takes all dimensions at once, and converges them to a single distance for each data point comparison, building a single matrix D that is used for the final path distance computation. The matrix formula, compared to one dimensional case, shown in Equation 2.2.

$$D_{i,j} = \sum_{k=1}^{f} \left|A_{i,f} - B_{j,f}\right| + \min\left(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1}\right)$$

Eq. 2.2 Multidimensional time warping distance matrix calculation.
Where f is used to iterate over the additional dimensions in each time-series.

2.4. Classification methods

2.4.1. Machine Learning methods

In 1959, Arthur Samuel conceived the term "Machine Learning" as "field of study that gives computers the ability to learn without being explicitly programmed." A machine learning program is able to adjust and improve its decision-making from input data it is exposed to, without its code being altered. In a way, it is data-driven decision making. For scribble recognition, it would not be inappropriate to use supervised learning.

Supervised machine learning involves providing training input data combined with desired output data, so the machine learning algorithm can adjust the model to meet desired expectations. With enough training, a suitable algorithm can create a model that is able predict appropriate outputs for new input data. Supervised learning can be used to perform regression and to train a model to predict numerical values for given inputs, or it can be used to categorize data. R. Kellogg has already provided the categories for scribbles, what is left then, to create a model that can learn how to label them from a dataset.

2.4.2. K Nearest Neighbours with DTW distance

While the KNN algorithm does not process nor train on input data, a labelled data set is still necessary. The KNN method is to compare an input to all samples of an available labelled dataset, and pick the k number of closest samples. Closeness or distance can be determined by an arbitrary function or algorithm, in this case, the Dynamic Time Warping algorithm. The final label for the new input would be that which appears most frequently in the K number of nearest neighbours, though the voting-selection function can also be arbitrary[3,14].

2.4.3. Support Vector Machines

A SVM classifier will attempt to discriminate classes, by treating them as points in multidimensional space, where the number of dimensions is equal to the number of features given to the algorithm. Then, the algorithm will attempt to calculate hyperplanes that separate input points by classes in said multidimensional space. Intuitively, in 2D space, a hyperplane is a line, in 3D space, a hyperplane is a plane. Support vectors are data points of different classes, nearest to the hyperplane on each boundary side. Hyperplanes are calculated during training, in such a way as to maximize the distance from the face of the hyperplane to its support vectors. In essence, training SVM on labelled data allows it to angle the hyperplane such that it can balance the distance for each support vector, maximizing the boundary[4].

2.4.4. Decision Trees

Conceptually, a decision tree is a branching structure of questions or conditions. The conditions with the most significant impact are at the top, and the most impactful is the root decision. Each decision splits the input data in half, repeating until there are no more splits to be made.

Similarly, in machine learning, a tree will split the features of its input based on conditions that it learns, until the final nodes are the classification answers for categorical tasks or real values for regression. Splits are chosen in such a way as to minimize prediction error or standard deviation in the two new halves. Trees are built using training data, and if unconstrained, will continue to split until it achieves 100% accuracy, however this will often lead to severe overfitting as the tree will create splits based on very small and specific differences in the training dataset that will not transfer to the general case. Trees are constrained by setting the maximum depth (decision trees are conventionally presented as growing from top to bottom) or pruning. Pruning removes the branches with low or no influence on prediction results, reducing the tree complexity and overfitting.

2.4.5. Random Forest

As an ensemble method, random forest uses the combined results of multiple decision trees to improve overall accuracy and reduce overfitting. Feature bagging is used to ensure that individual trees grow significantly different from each other. This process subsamples the original feature set, distributing the new subsets to each tree. If individual trees were not forced to deviate from each other in this way, their combined errors would be the random variation of the same set and in the end average out to no benefit. The basic idea is to create a forest of many shallow trees and combine their results, circumventing the overfitting problem of one complex tree[5].

2.4.6. Artificial Neural Networks

In vague resemblance to a human brain, artificial neural networks pass information through interconnected matrices of scaling weights and biases to transform input into desired output. These matrices or layers represent the network’s generalized knowledge[15]. At first, a network will be inept at any task, but given enough labelled data, an ANN can learn from its errors and appropriately adjust the values that represent the strength of individual connections. This process is called backpropagation, as the error feedback travels back up the network, in an attempt to minimize it for the next incoming input[21]. In this manner, an Artificial Neural Network can approximate rules that tie an arbitrary set of inputs to an arbitrary set of expected outputs. A classical ANN has one neuron layer between its input and output, neural networks with multiple layers are called deep neural networks and their governing field of study is Deep Learning.

Neural networks generally require large labelled training datasets in order to make accurate predictions about unlabelled data. They are, however, able to autonomously infer features within a dataset, that define different classes, provided that the classes are distinct enough[27].

2.4.7. Convolutions and Convolutional Neural Networks

In signal and image processing, a convolution is the process of applying a filter in order to glean features from data, different filters are able to transform input data into different feature maps (for example, thresholds of edges).

The same concept is applied in convolutional neural networks, however the filters are learned by the network and not predefined for specific features. Stacking multiple such operations, a network can learn the features of previous feature maps and create an abstracted representation of the input space at different levels of complexity[10].

Although digitized scribbles are recorded as “X” and “Y” signals, and saved as coordinate vectors for each axis, convolutional networks are still applicable. The signals are still geometrically related, and can be thought of as a more condensed representation of a 2D image. If the scribble vectors were recreated as images from saved data, the interpolation between data points would essentially be two dimensional padding. So these signals can be thought of as condensed image information.

The “X” and “Y” vectors can be concatenated together into a two pixel wide (or tall) image, and thus processed by a traditional convolutional network. In fact, each vector could be processed individually by a 1D convolution. Convolution layers are good at finding structure and data correlations within their windows, which is appropriate use for drawing signals when it’s necessary to find structure within them[19]. Using convolutional layers, it is possible for a network to learn the features of scribbles shapes, without relying on actual image data.

3. METHODOLOGY

3.1. Scribble dataset

Multiple scribbles are hand-drawn within the drawing process recorder software, one class at a time, they are batch exported into a labelled folder. The process is repeated for all twenty classes. Counts of samples in each scribble class are shown in Figure 3.1.

Each scribble example is stored in its own CSV (comma separated value) file, where each line represents a sample point in chronological order. Important columns in each row is the coordinate on the X axis and the Y axis for each sample in time.

Table 3.1. Number of scribble examples in each class.

Class	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
#	751	1116	527	1025	448	257	205	266	255	237

Class	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
#	132	119	190	151	201	103	300	265	247	242

3.2. Scribble data pre-processing

Data engineering is done in the Python programming language[25] with the following steps:

First, the dataset is loaded, inferring the class of each scribble based on its parent folder.

Then, the coordinates for the top-left corner of a scribble's bounding box is found. This point is subtracted from the scribble coordinate vector element-wise, so the stroke is shifted from global canvas space, to its own local space. In canvas space, the top left corner is (0, 0), which is where the shifted scribble now appears.

Next, the width and height of are computed for the scribble's bounding box. Then the geometric centre of this box is found. This point is then used to again subtract from the coordinate vector element-wise, the result is a scribble centred on the point (0, 0) in its own local space.

The largest dimension (X or Y axis) of the bounding box is found, and used to normalize all vectors within the range (-1, 1), both "X" and "Y" vectors are normalized by the maximum size in a single dimension, in order to keep the scribble's aspect ratio intact. Example of input and output of this process is shown in Figure 3.1.

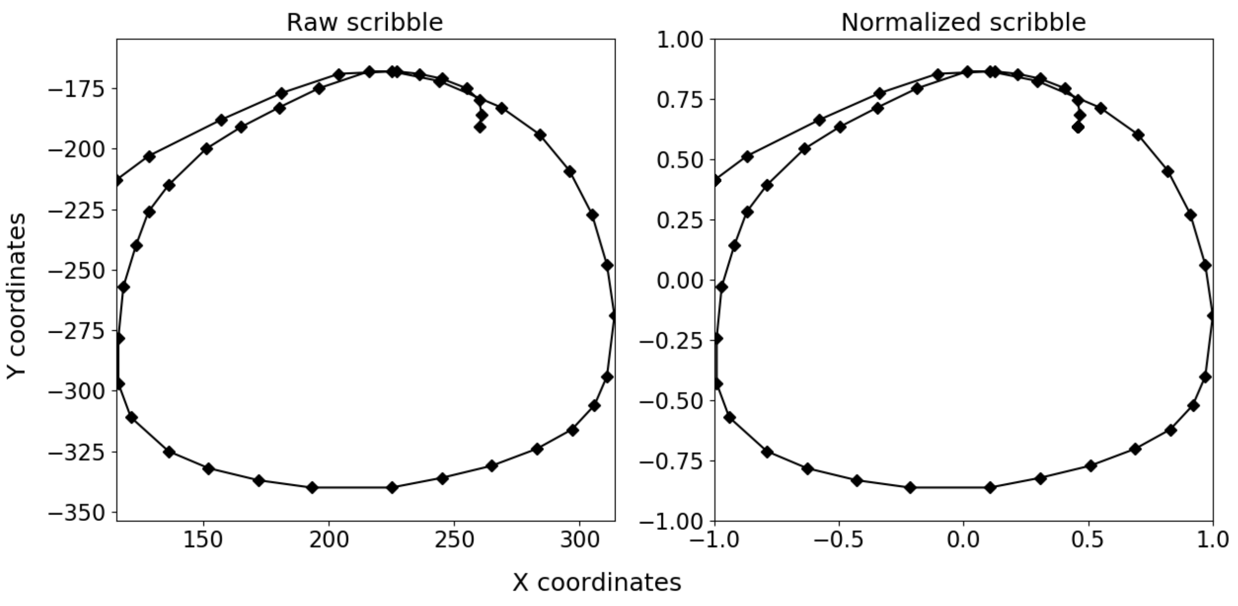


Fig. 3.1. Scribble in coordinate space before and after normalization.

To perform machine learning, it is generally much easier to work with samples of fixed size. Therefore, padding and truncation is performed in order to normalize sample sizes. 500 is chosen as the vector length, to fit most samples in the training set, as can be seen in Figure 3.2. Padding is performed for both beginning and end of coordinate vectors, with original data remaining in the middle of the vector. This is mostly for the sake of truncation, so if a complex scribble were to lose data points, the deformation would be uniform between the beginning and end of scribble. In some cases, a biased truncation could deform a scribble from one class into another. In Figure 3.3, a class 16 sample is shown to become like class 17.

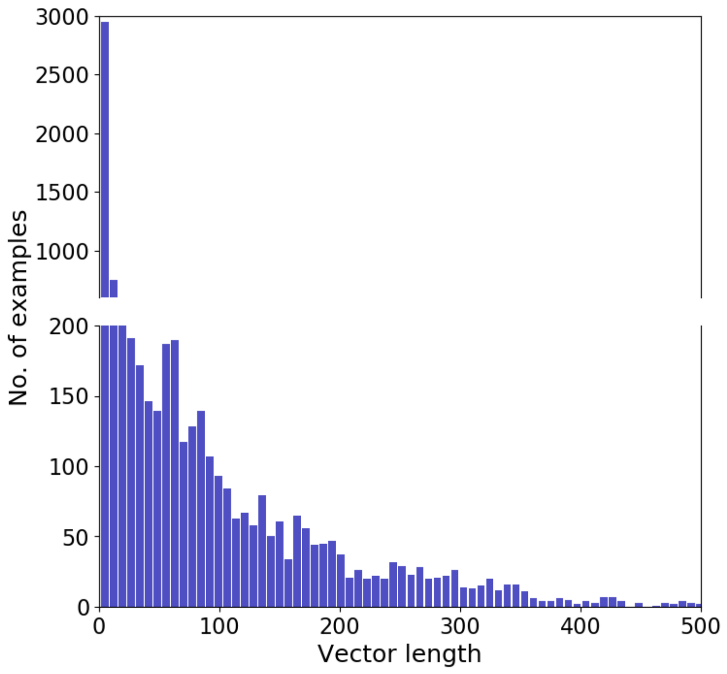


Fig. 3.2. Distribution of scribble instances, by the number of data points.

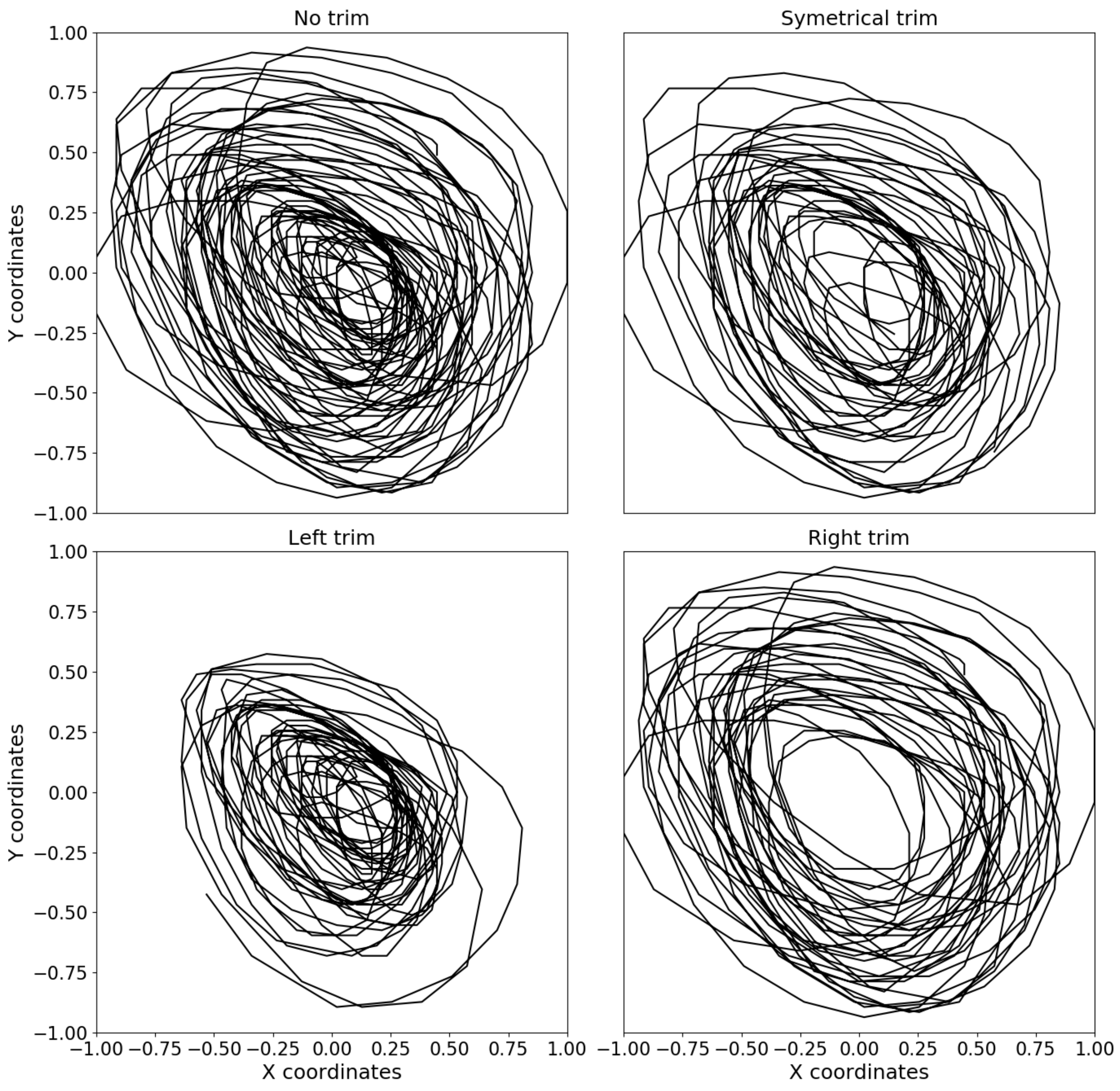


Fig. 3.3. Effects of different truncation schemes.

Coordinate vectors are padded by same adjacent values, as 0 values could geometrically link the beginning and end of a scribble to its centre-point, creating additional deformation, as seen in Figure 3.4.

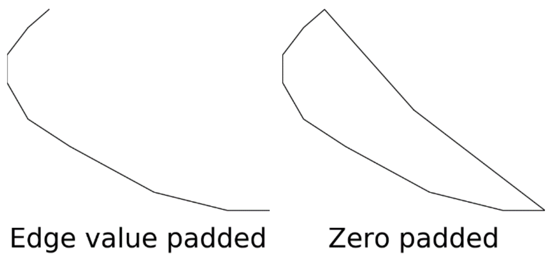


Fig. 3.4. Effects of different padding schemes.

As an additional feature, “angle” or the “A” vector is extracted to complement the “X” and “Y” vectors. The angle between each neighbouring coordinate point pair in the stroke is calculated, as illustrated in Figure 3.5. This way, the angle vector describes the curvature of the scribble. The benefits are dimensionality reduction and discarding information for size, and location, parameters that should be irrelevant to scribble classification by R. Kellogg’s classes. Furthermore, the angle vector has potential for feature engineering, for example, counting how many times the scribble performs a loop, whether it is generally curving smoothly or contains sharp turns.

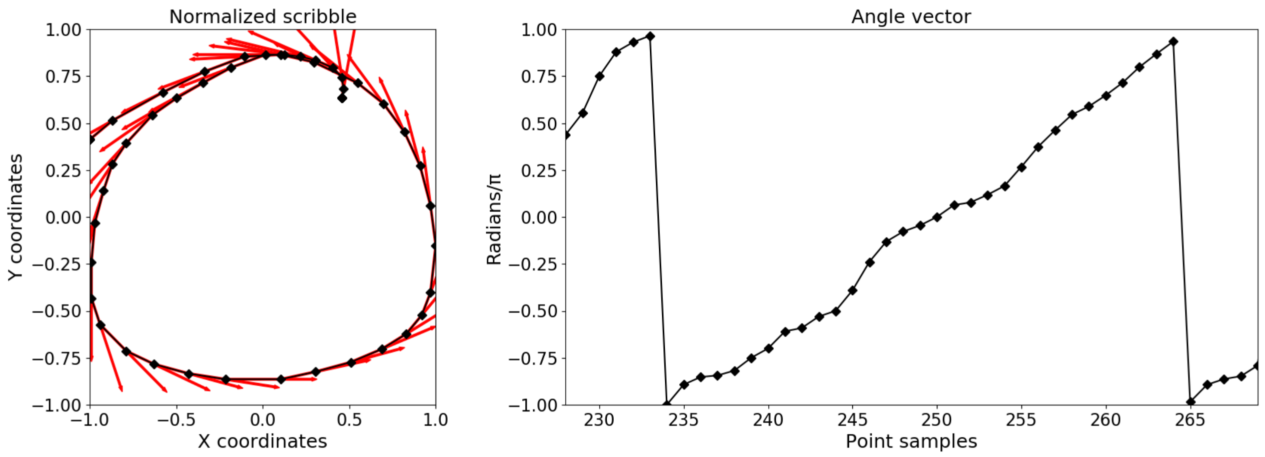


Fig. 3.5. Illustration of angle vector. Red lines represent the angle between neighbouring points, each angle is within -1 and +1.

3.3. Scribble data augmentation

To combat overfitting encountered using machine learning methods, data augmentation was performed.

Two different sets of augmentation operations are used, tailored to different groups within R. Kellogg’s classes.

Group one is sensitive to rotation, as it is basically the same scribble, but classified separately according to its angle (classes 2, 3, 4, 6, 7, 8). Group two contains classes where a scribble can be freely rotated and remain within its original class (classes 1, 5, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20).

Every transformation is preceded by a potential axis mirroring action. For group one, the mirroring options are: no mirror, X axis mirror, Y axis mirror, both axis mirror. For group two: no mirror, X axis mirror, Y axis mirror. With free rotation, mirroring will result in duplicate samples if the angles align, so both axis mirroring is redundant. Meanwhile, group one is restricted to only slight rotations, or else the sample will need to be give a new true label.

After mirroring, for each mirror option, group two samples are given a full rotation 0 to 360 degrees, for each degree step. Group one samples are rotated according to a list of small angles (for example 5, 3, -3, 5) in both directions, to keep the original scribble inclination more or less intact.

Using these steps, the original data set of 7037 samples is supplemented with an additional 148,029 samples derived from the original set.

3.4. Evaluation metrics

Primary method of evaluation is an accuracy score. This is implemented by scikit-learn[24] python package; for multi-label classification, this is the Jaccard index, where the input A is the set of predicted labels and B is the set of true labels.

$$\boldsymbol{J(A,B) = \frac{|A \cap B|}{|A \cup B|}}$$

Eq. 3.1. Jaccard index formula.

Training, evaluation and other time measures are taken with the built in python *time* package[25].

3.5. Methods and algorithms for classification

3.5.1. K Nearest Neighbour with Dynamic Time Warping

Nearest Neighbour method with Dynamic Time Warping as a distance measure is a popular approach for time-bound data classification.

For method evaluation purposes, a similarity matrix is computed. First, a $\frac{1}{3}$ test-$\frac{2}{3}$ train split is made, then each example in the training data is compared to each example in the test data. From this matrix, each scribble example has an arbitrary K number of nearest neighbours. To keep classes balanced, 500 examples from each class in the augmented dataset are taken. Tests are done with the angle vector and independent multidimensional dynamic time warping. Neither padding nor truncation is performed, as the DTW algorithm will readily accept variable length vector input, and empirically, padded vectors take longer to process. The classification results for variable K number of nearest neighbours shown in Figure 3.6.

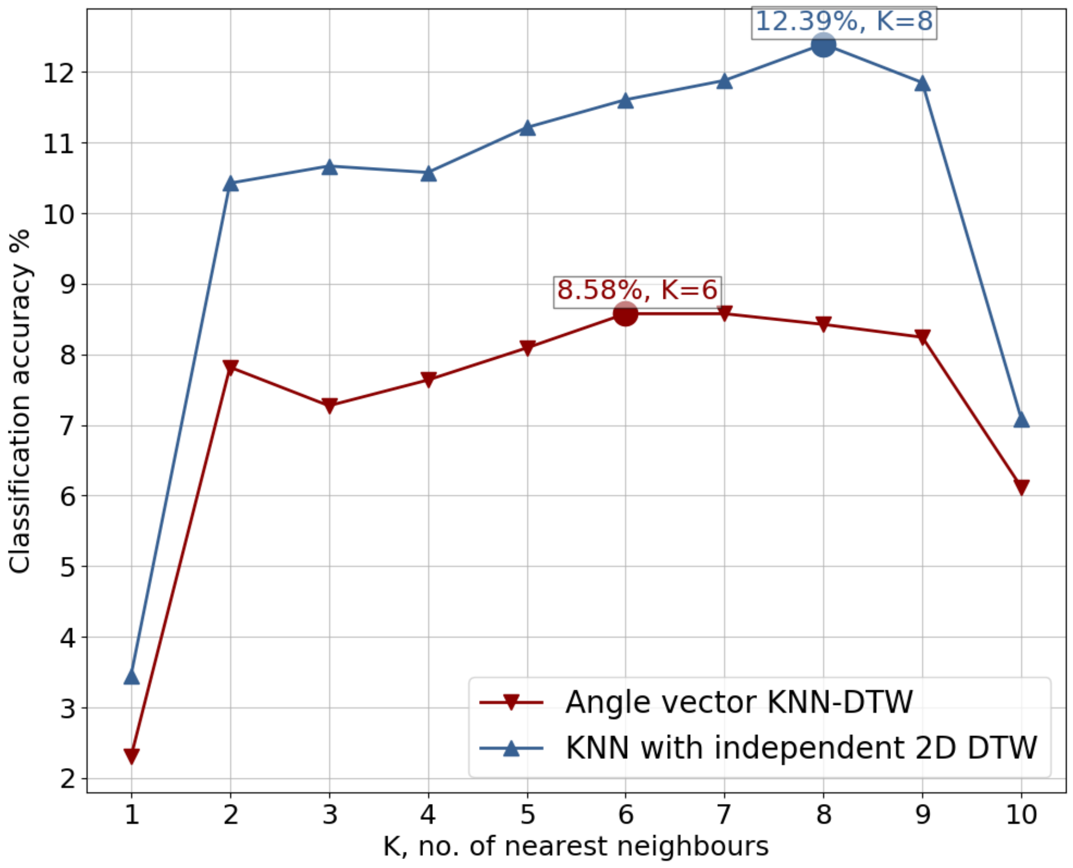


Fig. 3.6. KNN-DTW classification accuracy.

.

The reason for poor classification can be seen by examining a self-similarity matrix of the dataset. In an ideal situation, the matrix would show a distinct diagonal pattern, where classes of the same label are compared, such as in this synthetic toy example Figure 3.7.

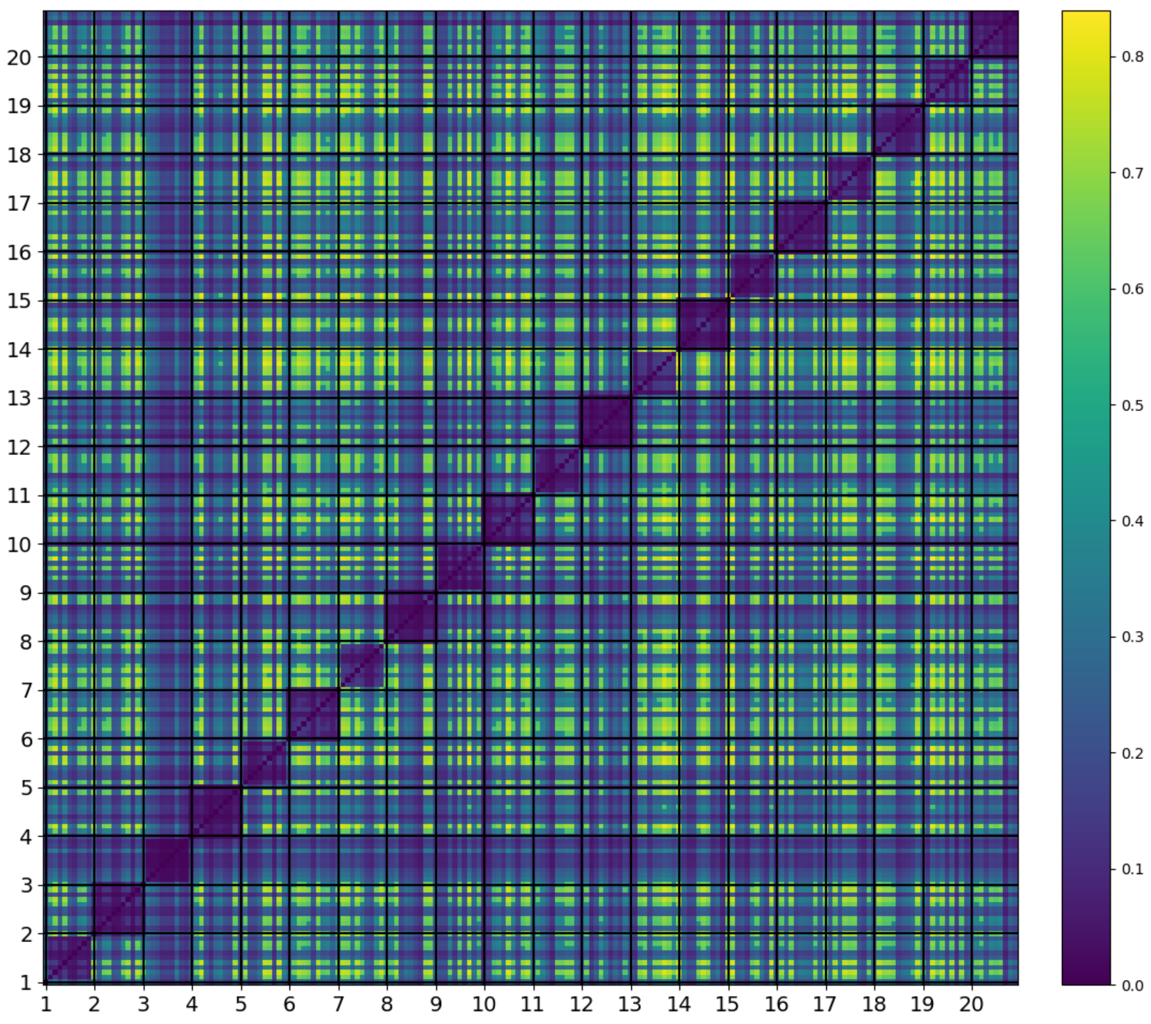


Fig. 3.7. Synthetic ideal self-similarity matrix. Axis are class labels. E.g. between 1 and 2 are class 1 samples, 20 and above are all class 20 samples.

In practice, however, the self-similarity matrix of the scribble dataset does not contain such convenient similarities, as shown in Figure 3.8.

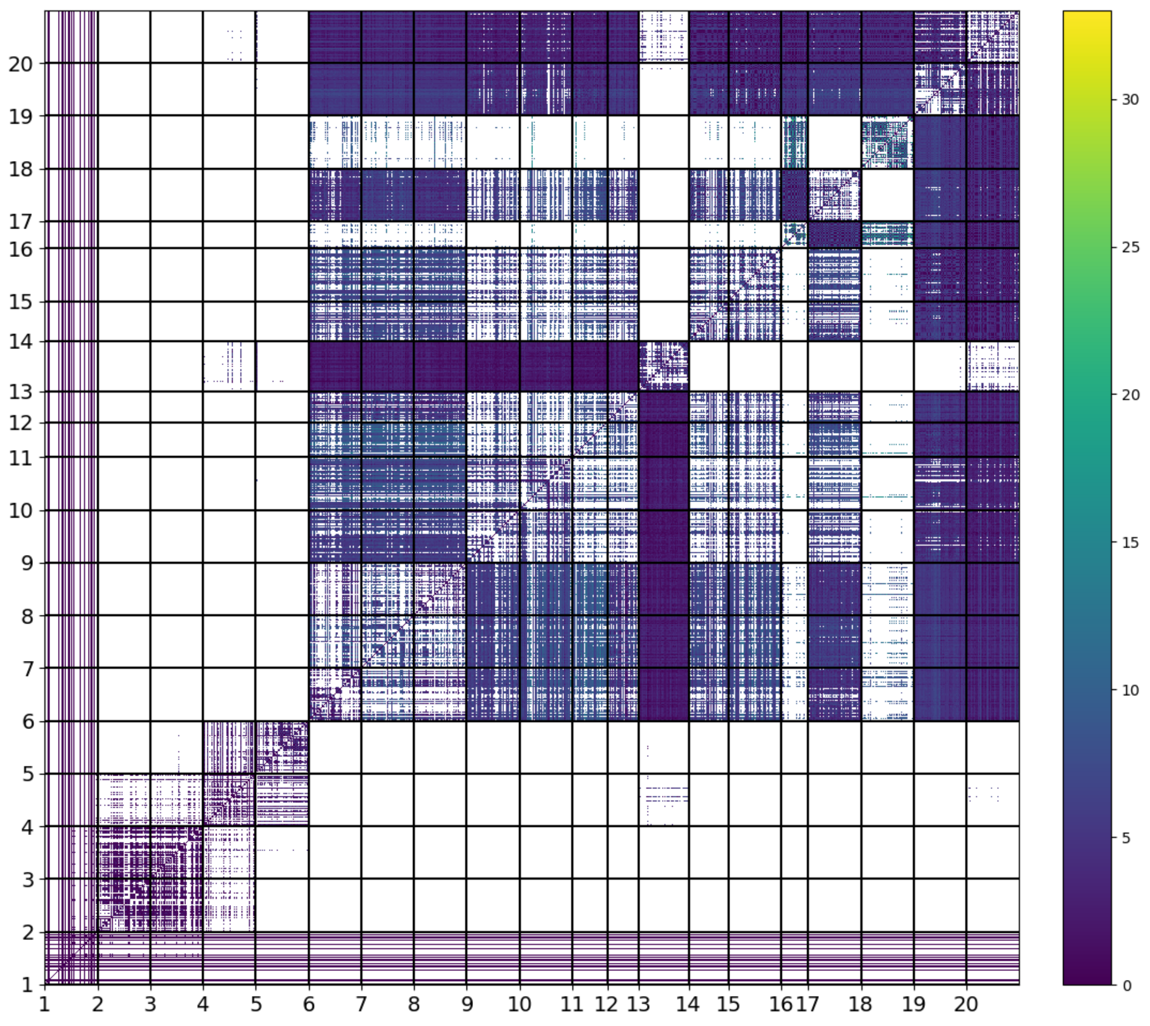


Fig. 3.8. Real self-similarity matrix of scribbles categorized by R. Kellogg's classes. Axis are class labels, same as Fig 3.7.

White areas are distance measures that the optimized algorithm ended prematurely and evaluated as pseudo-infinity (in this case, 1e10). It can be seen, that there is little obvious self-similarity between the classes, and a lot of inter-class confusion. With a nearest neighbour method, class 1 requires special treatment, as it readily matches as similar with all other classes. Other, simple one line classes (2, 3, 4 ,5) have either been optimized away or match their adjacent classes. More complex classes have variable relations with other complex classes, but show little in-class similarity between examples.

By examining self-similarity within individual classes (Figures 3.9, 3.11), it can be seen why scribble classes as defined by R. Kellogg, are not well categorized by the dynamic time warping distance. Self-similarity matrices are computed by the FastDTW[28,32] algorithm to avoid calculations omitted by optimization and increase visibility.

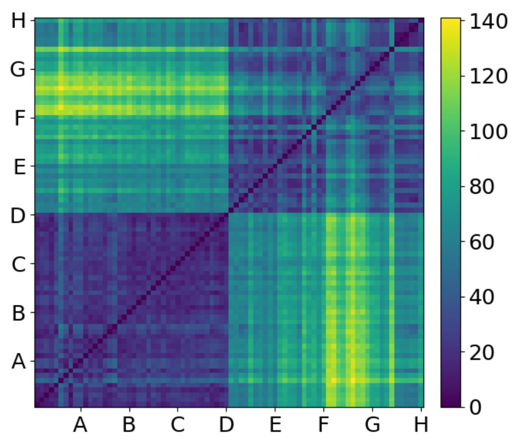


Fig. 3.9. Self-similarity matrix of different class 8 variations.

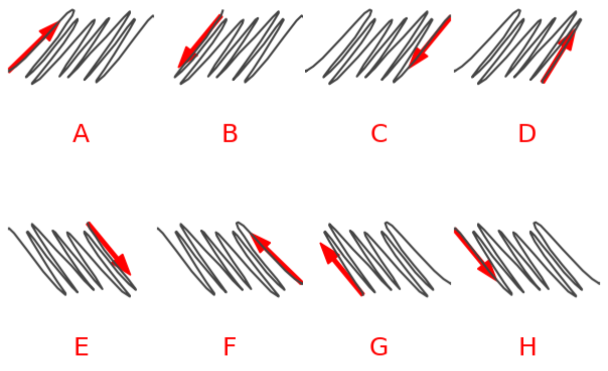


Fig. 3.10. Different ways to render class 8.

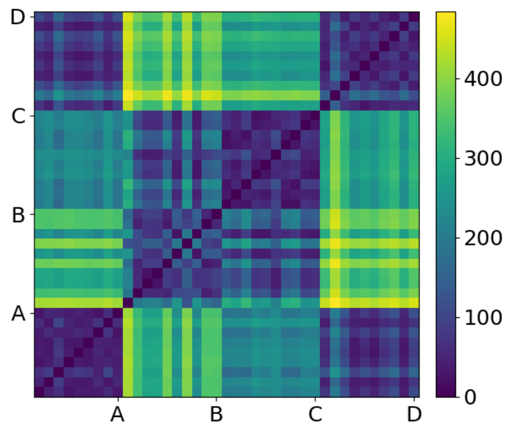


Fig. 3.11. Self-similarity matrix of different class 16 variations.

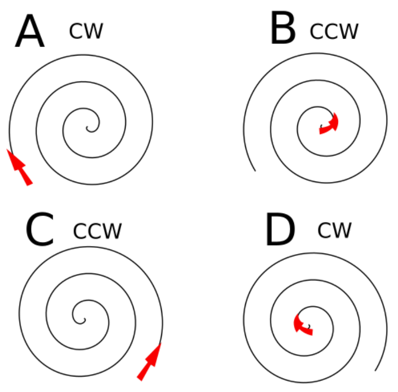


Fig. 3.12. Ways to draw class 16. While this class is not a spiral, this is the trend of the pen motion as it is rendered.

Within R. Kellogg's classes, most scribbles can be rendered in at least several different ways (Figure 3.10, 3.12) that the dynamic time warping algorithm is sensitive to. The same time-series, mirrored geometrically, will yield large distance, or in other words, high dissimilarity. The problem is more severe if DTW distance is calculated by separate "X", "Y" vectors instead of the angle vector – the inter-similarity between variations decreases, looking more like an ideal self-similarity matrix, except in this case that is disadvantageous.

To use this method for scribble classification, R. Kellogg's classes would need to be redefined, some merged, some split, and perhaps further post-processing could transform the results to match the original classes.

Furthermore, the average length of a scribble class is moderately correlated (0.675 correlation coefficient) with the average dynamic time distance measure of that class. Shorter scribbles are expected to be erroneously chosen more often as the nearest neighbour.

3.5.2. Support Vector Machines

The scikit-learn[24] python package was used to implement a SVM classifier. It was given the same train-test split for training and evaluation as in other tests. The input was the angle vector (as an alternative to the “X” and “Y” vectors), in order to have reduced dimensionality.

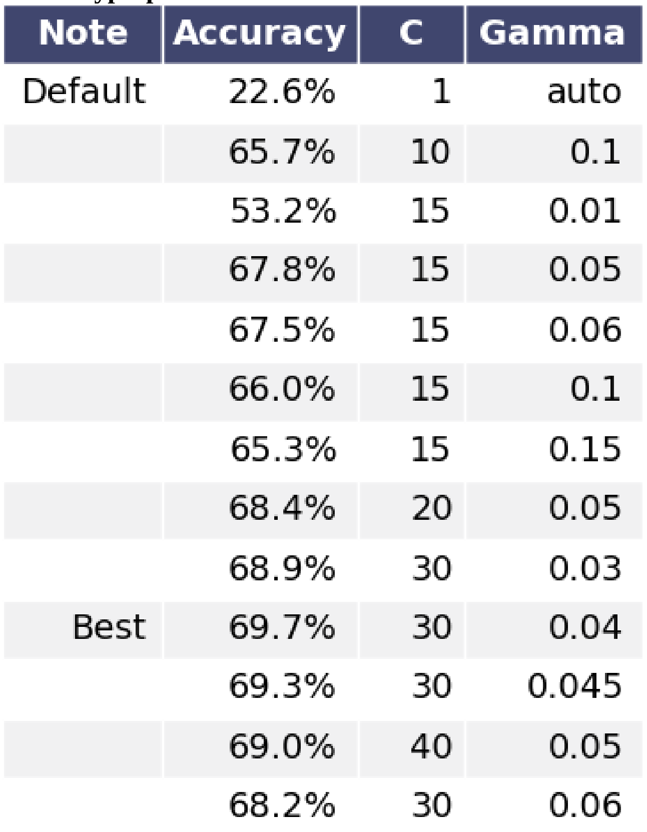
This SVM implementation has several hyperparameters that can be tuned. A search for good hyperparameters was done by creating sets of parameter options then running the classifier through each permutation, with cross validation for evaluation. Different hyperparameter selections are shown in Table 3.2. After final tweaks with individual adjustments, the chosen parameters with most influence to evaluation results were:

C (soft margin parameter): 30

Gamma (RBF kernel param.): 0.04

Choosing these hyperparameters yielded significant gains over the default settings, and evaluated with the base dataset (4714 training, 2323 test data), the classifier produced about 69.7% accuracy. Training and testing with the same parameters on the augmented dataset (99179 training, 48850 test data) yields 83% accuracy.

Table 3.2. Hyperparameter selection influence on SVM accuracy



Note	Accuracy	C	Gamma
Default	22.6%	1	auto
	65.7%	10	0.1
	53.2%	15	0.01
	67.8%	15	0.05
	67.5%	15	0.06
	66.0%	15	0.1
	65.3%	15	0.15
	68.4%	20	0.05
	68.9%	30	0.03
Best	69.7%	30	0.04
	69.3%	30	0.045
	69.0%	40	0.05
	68.2%	30	0.06

3.5.3. Random Forest

A random forest classifier was implemented the same way as with SVM – through the scikit learn[24] Python package. Input was identical to what SVM received.

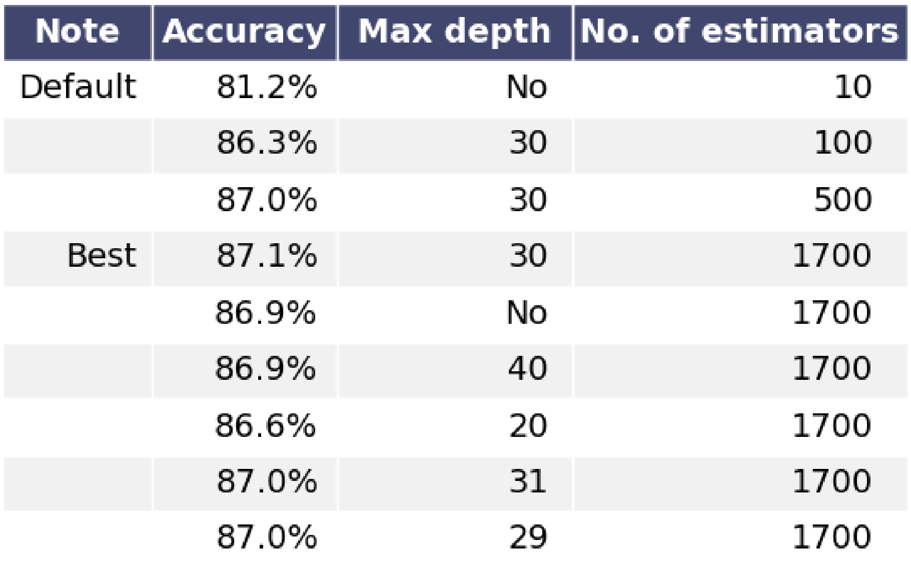
Search for good hyperparameter values was done similarly as with SVM, except the parameter permutations were chosen and tested randomly, instead of exhaustively, to reduce the amount of calculations. Parameters were first tested with subsampled base dataset, 10 samples per class, then 50 samples per class, until the available parameter choices were trimmed and finally evaluated through with the full dataset. Parameters with most influence over the results are shown in Table 3.3. Their final values are:

Max depth: 30

No. of estimators (trees): 1700

The final accuracy for the base dataset (same split as SVM) is 87.1%. With the augmented dataset, accuracy is 94.5%

Table 3.3. Hyperparameter selection influence on Random Forest accuracy.



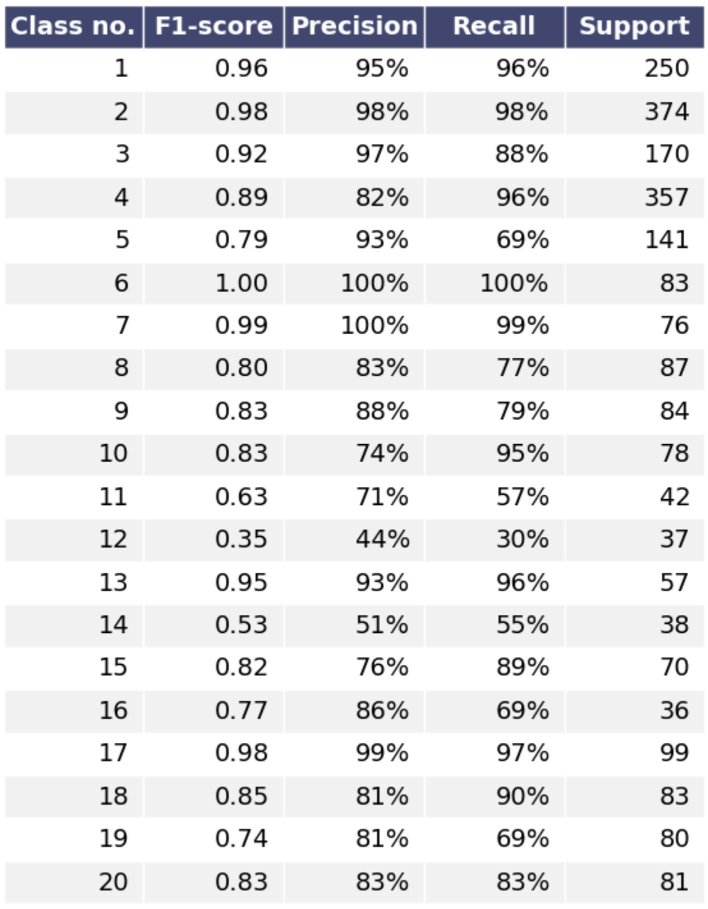
Note	Accuracy	Max depth	No. of estimators
Default	81.2%	No	10
	86.3%	30	100
	87.0%	30	500
Best	87.1%	30	1700
	86.9%	No	1700
	86.9%	40	1700
	86.6%	20	1700
	87.0%	31	1700
	87.0%	29	1700

3.5.4. Convolutional networks

Keras[6] with Tensorflow[1] backend was used to quickly prepare and test neural network models, processing on a GPU for speed.

A CNN was not the first choice for scribble classification, as recurrent neural networks (RNN) are traditionally used to support variable length input. However, padding was inevitable, as the scribble vectors can be as short as a single point of data. If padding is embraced to have scribble vectors at consistent length, then it is trivial to use non recurrent network architectures. After forming a 500 by 2 input image set from scribble coordinate vectors, a simple CNN was first to be tested, its architecture is shown in Figure 3.13, the accuracy for the base dataset is 88% with classification report shown in Tabke 3.4.

Table 3.4. Classification report of basic CNN



Class no.	F1-score	Precision	Recall	Support
1	0.96	95%	96%	250
2	0.98	98%	98%	374
3	0.92	97%	88%	170
4	0.89	82%	96%	357
5	0.79	93%	69%	141
6	1.00	100%	100%	83
7	0.99	100%	99%	76
8	0.80	83%	77%	87
9	0.83	88%	79%	84
10	0.83	74%	95%	78
11	0.63	71%	57%	42
12	0.35	44%	30%	37
13	0.95	93%	96%	57
14	0.53	51%	55%	38
15	0.82	76%	89%	70
16	0.77	86%	69%	36
17	0.98	99%	97%	99
18	0.85	81%	90%	83
19	0.74	81%	69%	80
20	0.83	83%	83%	81

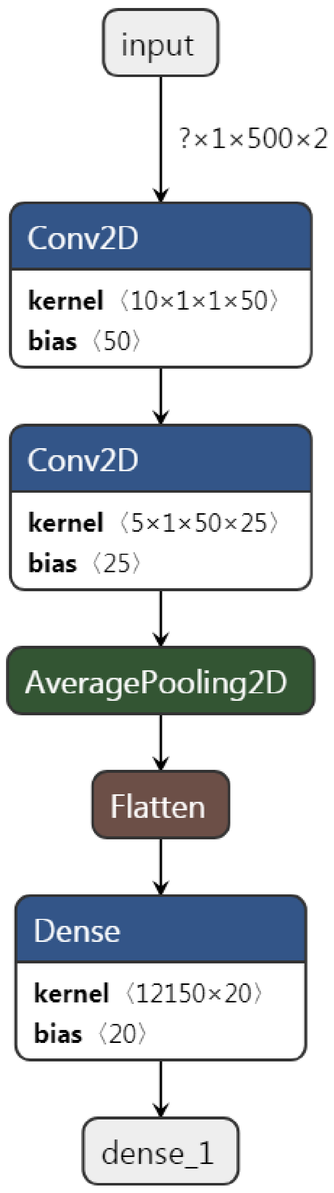


Fig. 3.13. Basic generic CNN architecture.

Because scribble data is spatial and adjacent values represent connections, padding is done by copying the first and last value in the vector, this way, no additional geometric structure should be introduced. As padding did not seem to cause any obvious issue discriminating between originally short and originally long scribble sequences, using a neural network is worth further investigation.

By making small changes to the network and testing for accuracy improvements, it's been determined that a pooling layer is unnecessary, and accuracy increases significantly when a layer with convolution window height 2 is introduced. Previously, each "X" and "Y" vector have been parsed by the network individually, then merged only in the final dense layer. The improved network (Figure 3.14) yielded validation accuracy of 93%. Model confusion matrix in Figure 3.15.

Overfitting was observed, despite various alterations to network configuration with insignificantly differences in results. Thus, the augmented dataset was used instead to counter overfitting, producing 98% accuracy; confusion matrix is shown in Figure 3.17.

In all tests, datasets were split, $\frac{2}{3}$ for training and $\frac{1}{3}$ for evaluation. All models were trained with *adam* optimizer and cross-entropy loss function.

Adapting the network to also take the angle vector in its input "image" (500x3) made no perceptible differences in classification accuracy.

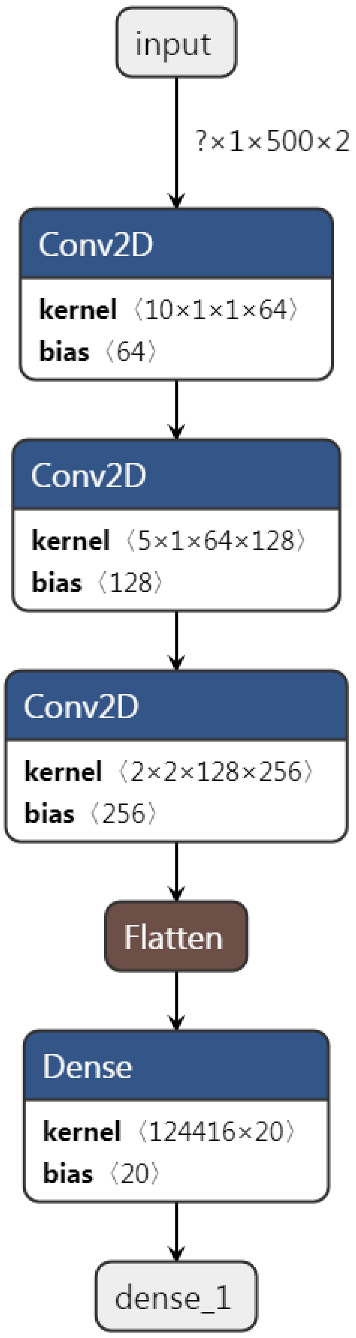
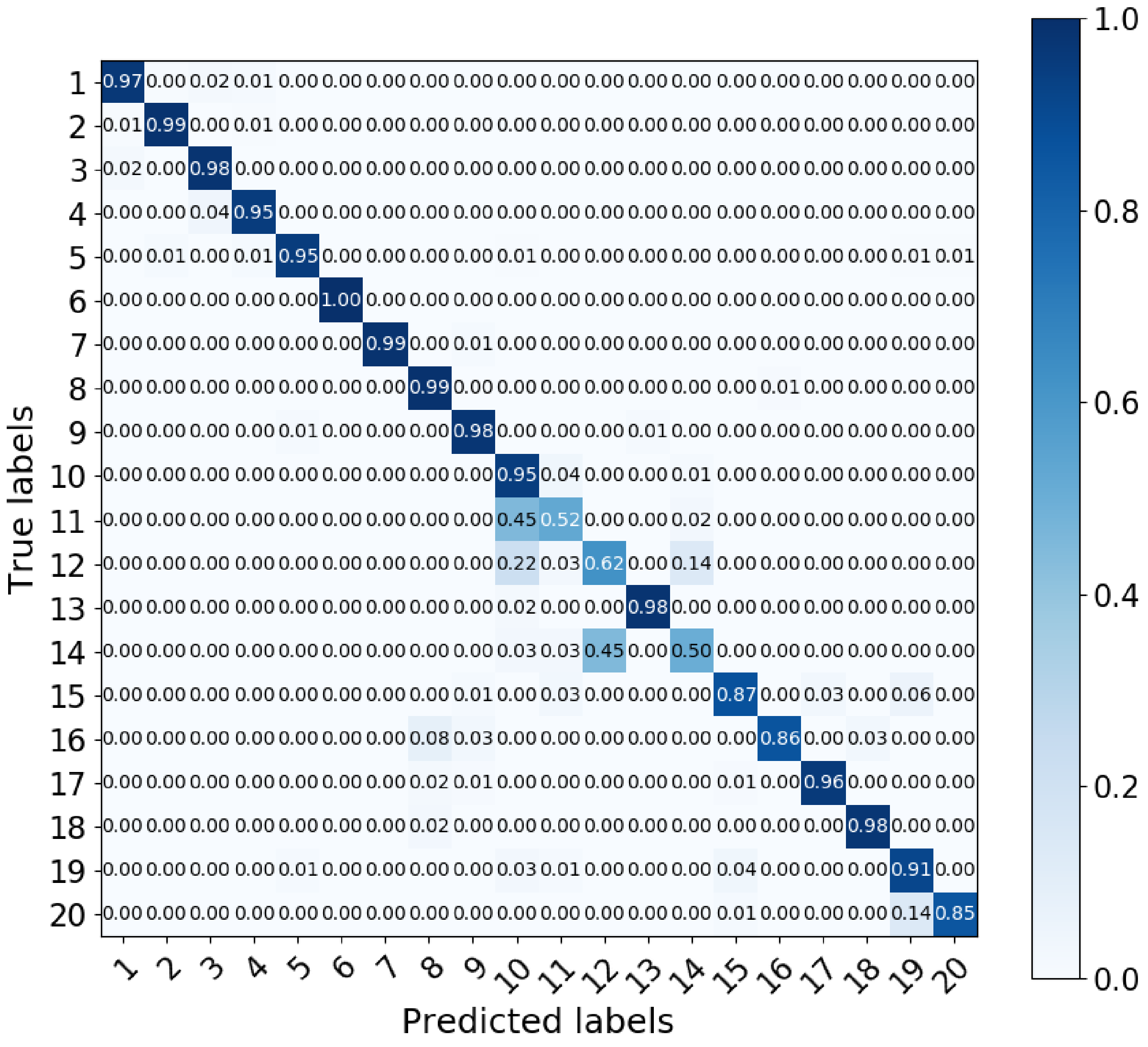


Fig. 3.14. CNN architecture tailored for scribble recognition.



True labels \ Predicted labels	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.97	0.00	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.01	0.99	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.02	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.04	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.01	0.00	0.01	0.95	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
6	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.45	0.52	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.22	0.03	0.62	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.45	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.03	0.00	0.00	0.00	0.87	0.00	0.03	0.00	0.06	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.00	0.03	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.96	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00
19	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.03	0.01	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.91	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.14	0.85

Fig. 3.15. Confusion matrix for tailored CNN trained and evaluated on base dataset.

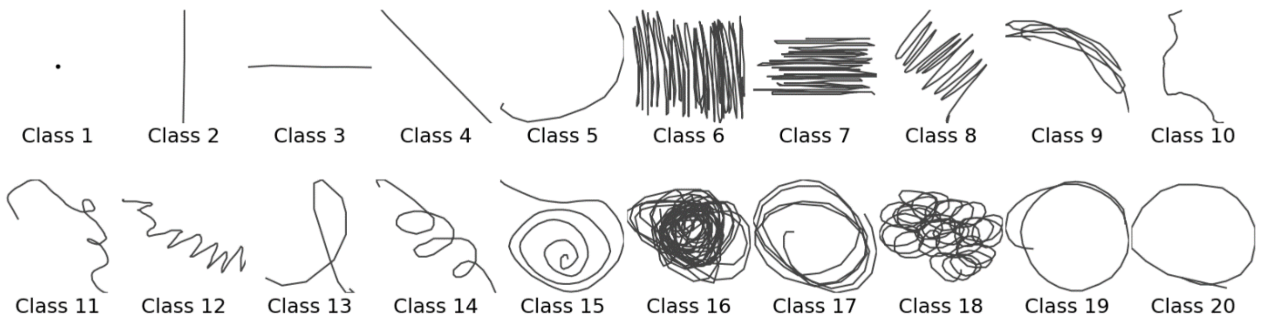
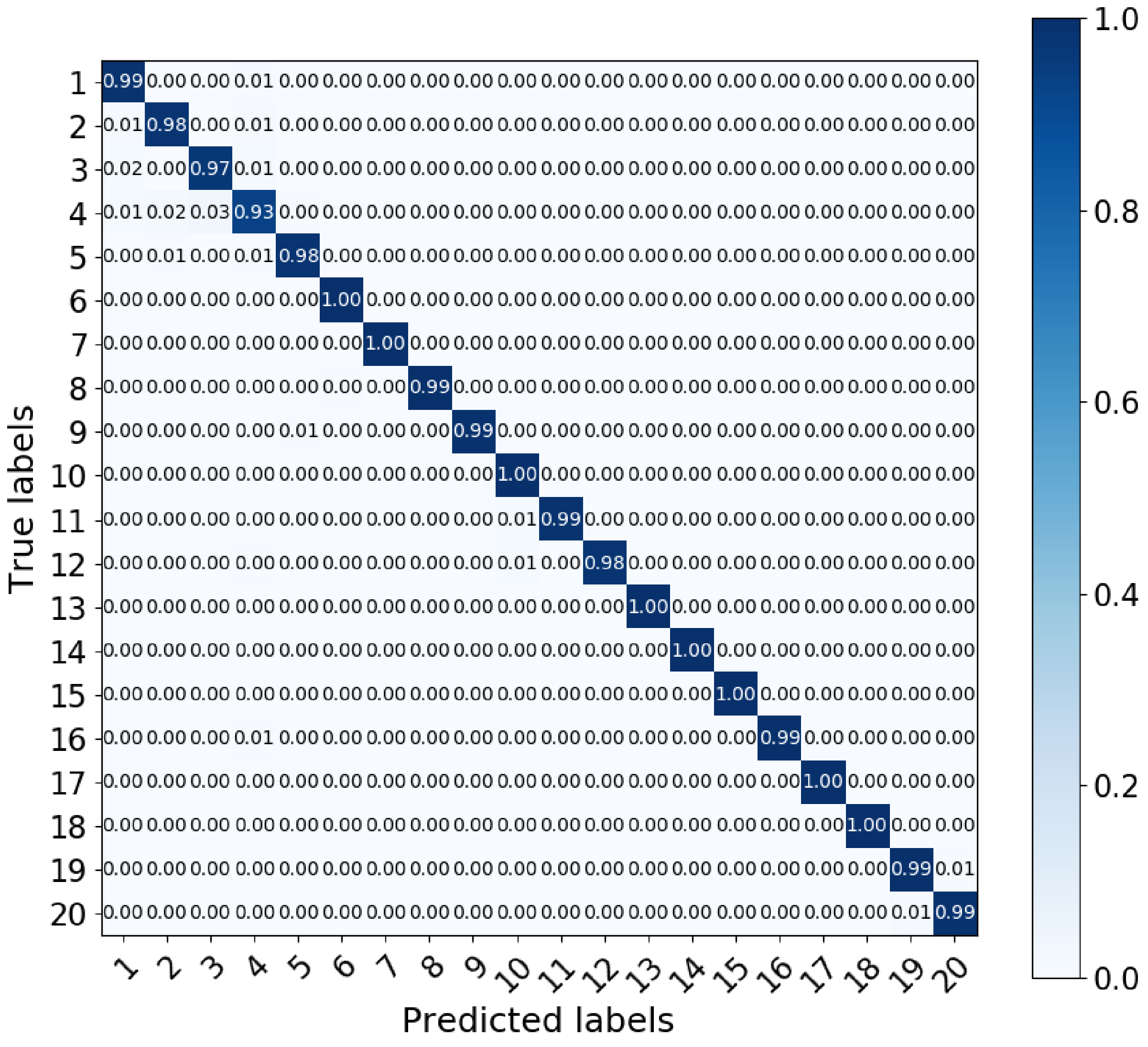


Fig. 3.16. Visual representation of classification labels.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.99	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.01	0.98	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.02	0.00	0.97	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.01	0.02	0.03	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.01	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.01
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.99

Fig. 3.17. Confusion matrix for tailored CNN trained and evaluated on augmented dataset.

3.6. Method comparison

3.6.1. Accuracy

Best accuracy achieved by each model is shown in Table 3.5.

Table 3.5. Method accuracy comparison.

Classification method	Accuracy
KNN IM DTW	12.38%
KNN A DTW	8.58%
SVM	83.06%
R Forest	94.59%
CNN	98.5%

K Nearest Neighbour classifier with DTW distance measure is a popular choice for time-series classification, but is incompatible with scribble classification using R. Kellogg's classes. Simply, these classes have feature differences between each other that DTW is incompatible with for comparison.

Vectors with 500 elements is not ideal input for either SVM nor Random Forest classifiers, as each element becomes a feature dimension. The vector length is necessary, to not trim longer scribble instances, however, and it is still much more condensed data than a scribble image would've been, similar to an extracted contour[1619]. Despite the high dimensionality, with the large number of scribble examples in the augmented dataset, SVM and especially Random Forest methods achieve respectable accuracy.

Convolutional neural networks for time-series classification is not a new concept, and perhaps particularly fitting for scribble classification as the time series are still spatial data. Using data augmentation has practically eliminated overfitting, as training and validation accuracies are generally very close. The accuracy limit of this simple network has been reached, could perhaps be improved only by inspecting the base dataset for any erroneous samples and tuning augmentation parameters as in some cases rotation during augmentation could produce samples that ought to be under a different class post-augmentation.

Other neural network methods that could learn the relevant features of R. Kellogg's classes from the given dataset are also likely to be very successful, however a CNN has proven to be very easy to implement while giving good results.

Recurrent networks were considered because of varying scribble sample vector lengths, however a constant size window is still required to travel over each data sample, most of which are not very long and some (in class 1) are just a single point, thus a padding scheme was mandatory and when all vector lengths are equalized, simpler networks than RNN are much easier to implement.

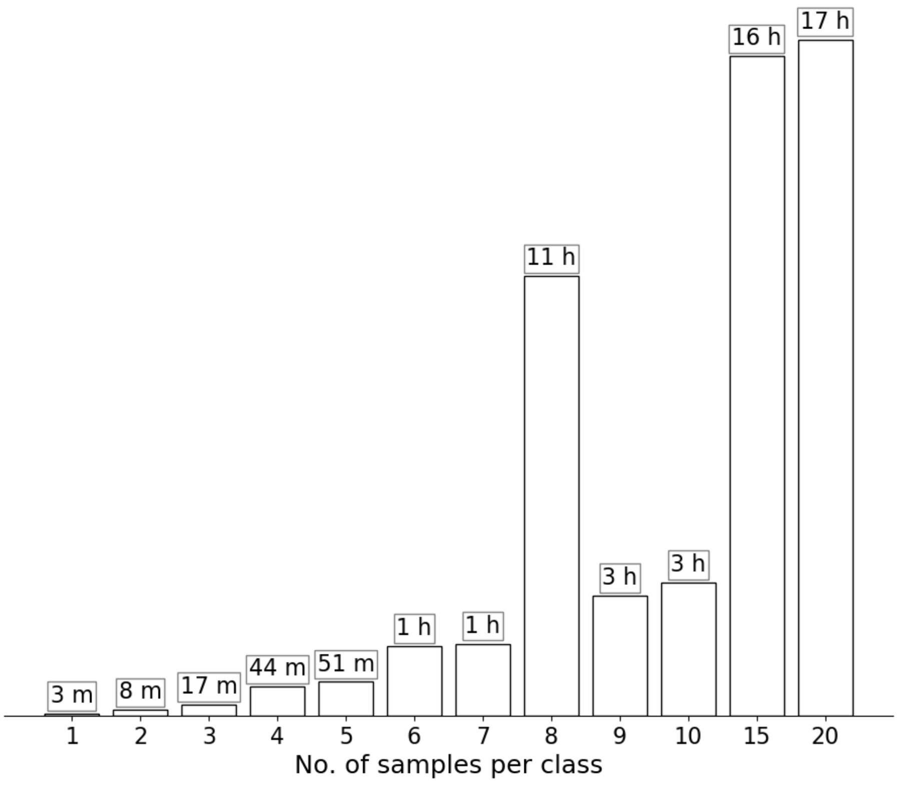
3.6.2. Time

For nearest neighbour classification with DTW, the most expensive operation is computing the similarity matrix between training and test data. Naïve DTW's time complexity is *O(N M)* where *N* and *M* are sequence lengths. Displayed times in Table 3.6 are for 500 samples per class, with 20 classes, making it 10,000 samples of scribbles. The train-test split separates them into 6700 training and 3300 evaluation samples, forming a 6700 by 3300 matrix, which is 22,110,000 DTW operations. Fortunately, the UCR-Suite[26] of optimizations for DTW exists, making the KNN DTW method evaluation possible. Independent multi-dimensional DTW requires two operations per sample, which is still done in very reasonable time. Dependent M-DTW was not evaluated as it could not be confirmed whether the UCR-Suite DTW package supports multidimensional operation directly, though classification accuracy is not expected to be much improved from independent M-DTW.

Table 3.6. Time to compute distance matrices with UCR-Suite DTW

Distance measure	Samples per class	Time
IM DTW	500	2:58 h
A DTW	500	2:05 h

To paint a picture how time-consuming DTW is without the UCR-Suite, time taken to compute self-similarity matrices as a function of samples per class shown in Figure 3.18. using FastDTW[28,32] pure python implementation. Inconsistencies are due to random sampling of the dataset for each subset of classes used in self-similarity matrix computation.



.

Fig. 3.18. Time dependence on number of samples for self-similarity matrix calculations with FastDTW.

Support Vector Machines and Random Forest classifiers share a similarity in that there are standard methods to search for good hyperparameters that are similar in process to training a model on a dataset, taking significant time, thus SVM and Random Forest classifier times are both in Figure 3.7. It is likely that high dimensionality has significantly hampered SVM classifier performance. It is notable that Random Forest is able to make predictions very quickly.

Table 3.7. Time to find hyperparameters for, train and evaluate SVM and Random Forest classifiers.

Classifier	Hyperparam. search time	Train samples	Test samples	Train time	Eval. time
SVM	30 m	4714	2323	6.5 s	4.3 s
SVM	-	99179	48850	30 m	25 m
R Forest	97 m	4714	2323	8.2 s	0.3 s
R Forest	-	99179	48850	18 m	37.7 s

When testing convolutional neural networks, the largest time expense is in training. Times shown in Figure 3.8. Empirically, training time depends on the size of the training dataset, number of epochs, and the number of parameters that the model is trying to learn. Training was done on a GeForce GTX 1070 Ti GPU.

Table 3.8. Time to train and evaluate CNN models.

Model	Training data	Parameters	Epochs	Train time	Eval. time
CNN0	4714	249845	19	19 s	1.4 s
CNN0	99179	249845	5	100 s	6.8 s
CNN1	4714	2661460	10	25 s	2.4 s
CNN1	99179	2661460	5	251 s	3.9 s

CNN0 is the first arbitrary example network while CNN1 is the final network tailored for scribble recognition accuracy, yet still very simple. These CNNs are able to process the training slice of the entire augmented dataset, perhaps a more complex CNN or different type of neural network with higher accuracy would take more time.

It is, however, unfair to directly compare the CNN times to other methods, as the Tensorflow backend for the model was always run on a graphics card, taking advantage of NVIDIA's CUDA technology, which is significantly faster than if the model was run on a CPU. Furthermore, CUDA implementations of other evaluated models exist[17][12][33] (but were not tested for implementation simplicity and tool compatibility).

4. DISCUSSION

Potential combinations or enhancements of tested methods are still unexplored. For example: feature reduction using neural networks before classification by Random Forest, cross-class relationships of DTW distances as classification feature. Entirely untried is the concept of compound scribbles (two classes made with a single stroke), that has potential solutions in adapting the CNN classifier for a sliding window approach. Nevertheless, without complicating the problem further, the current chosen best CNN classifier is difficult to defeat, especially considering its simplicity. For the particular problem of scribble recognition by R. Kellogg’s classes, CNNs or other neural networks deserve further efforts.

It would have been interesting to find scribble instances in the dataset which are most often misclassified, this could be an efficient method to verify whether all samples belong to correct classes, as there could be errors in initial input or too aggressive transformations during augmentation. Progress has been made creating auxiliary software for dataset inspection and preparations for classification as a result of this work.

The final classifier will need to be connected with the Drawing Recorder software to run real-time for practical purposes. Hopefully it might lead to new research into children scribbles in the field of psychology, in particular, early development, hand-eye coordination and motor functions.

5. CONCLUSIONS

A scribble dataset was created with an adjustable augmentation scheme that can multiply the dataset to a greater or lesser degree. After analysing the dataset and attempting practical nearest neighbour classification with dynamic time warping as the distance measure it was shown that the method is incompatible with R. Kellogg's scribble classes.

Support vector machine and random forest machine learning algorithms reach moderate performance, 83.06% and 94.59% accuracy respectively, on reduced dimensionality data (from points to angles). However, there is space for further feature engineering for dimensionality reduction on the time axis, which could lead to better generalizations.

Convolutional neural networks are found to be able to discriminate well between R. Kellogg's classes, and is chosen as the best classifier among the tested. Recorded scribbles vectorised as points in time seem to be fairly condensed and distinct representations considering the ease with which CNNs handle them. Though the tested network can predict scribbles with 98% accuracy, the model's architecture is very simple, and more complex, deeper layer arrangement, with enough tailoring, could achieve even better results.

6. REFERENCES

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, …, X. Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems.* 2015. Available at: https://tensorflow.org. [accessed: 2018-06-30].
2. R. Arnheim. *Art and Visual Perception: A Psychology of the Creative Eye*. University of California Press, 2004.
3. A. Bagnall, J. Lines. 2014. *An Experimental Evaluation of Nearest Neighbour Time Series Classification*. arXiv:1406.4757 [cs.LG]. Available at: https://arxiv.org/abs/1406.4757 [accessed: 2019-05-29].
4. B. E. Boser, I. M. Guyon, V. N. Vapnik. *A training algorithm for optimal margin classifiers*. Proceedings ACM Workshop on Computational learning theory, pp.144-152, 1992 USA. ISBN:0-89791-497-X.
5. L. Breiman. *Random Forests*. Machine Learning, v.45 n.1, pp.5-32, 2001 [doi>10.1023/A:1010933404324].
6. D. Falbel, J. J. Allaire, F. Chollet, Y. Tang, W. V. D. Bijl, M. Studer, S. Keydana. 2015. *Keras.* Available at: https://keras.io [accessed: 2019-04-24].
7. M. J. Fonseca, C. Pimentel, J. A. Jorge. *CALI: An Online Scribble Recognizer for Calligraphic Interfaces*. AAAI Spring Symposium on Sketch Understanding, pp.51-58, 2002.
8. M. J. Fonseca, J. A. Jorge. *Experimental evaluation of an on-line scribble recognizer*. Pattern Recognition Letters, v.22 n.12, pp.1311-1319, 2001.
9. A. Fornnés, J. Lladós, G. Sánchez, D. Karatzas. *Rotation invariant hand-drawn symbol recognition based on a dynamic time warping model*. International Journal on Document Analysis and Recognition v.13 n.3, pp.229-241, 2010.
10. K. Fukushima. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological Cybernetics v36 n4, pp.193-202, 1980.
11. V. Giedrimas. L. Vaitkevičius. A. Vaitkevičienė. *Drawing process recording tool for Eye-hand Coordination modeling*. International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 588-592, 2017. Electronic ISBN: 978-953-233-090-8.
12. H. Grahn, N. Lavesson, M. Hellborg Lapajne, D. Slat. *CudaRF: A CUDA-based implementation of random forests.* IEEE/ACS International Conference on Computer Systems and Applications, pp.95-101, 2011.

13. D. Ha, D. Eck. 2017. *A Neural Representation of Sketch Drawings*. arXiv:1704.03477 [cs.NE] Available at: https://arxiv.org/abs/1704.03477 [accessed: 2019-05-20].

14. G.A. ten Holt, M.J.T. Reinders, E.A. Hendriks. *Multi-Dimensional Dynamic Time Warping for Gesture Recognition*. Thirteenth annual conference of the Advanced School for Computing and Imaging 2007.

15. J. J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities*. National Academy of Sciences v.79 n.8, pp.2554-2558 1982. Print ISSN: 0027-8424.

16. S. Hou, K. Ramani. *Structure-oriented contour representation and matching for engineering shapes*. Computer-Aided Design v.40 n.1, pp.94-108, 2008.

17. C. Hundt, B. Schmidt, E. Schömer. *CUDA-Accelerated Alignment of Subsequences in Streamed Time Series Data*. 43rd International Conference on Parallel Processing, pp.10-19, 2014 USA. Print ISSN: 0190-3918.

18. R. Kellogg, *Analyzing Children's Art*, Girard & Stewart, 2015.

19. Y. LeCun, P. Haffner, L. Bottou, Y. Bengio.. *Object Recognition with Gradient-Based Learning*. Proceeding Shape, Contour and Grouping in Computer Vision, p.319, 1999. ISBN:3-540-66722-9.

20. W. Li, T. Hammond. *Using scribble gestures to enhance editing behaviors of sketch recognition systems*. Proceeding CHI '12 Extended Abstracts on Human Factors in Computing Systems, pp.2213-2218. 2012. ISBN: 978-1-4503-1016-1.

21. S. Linnainmaa. *Taylor expansion of the accumulated rounding error*. BIT Numerical Mathematics, v.16 n.2, pp.146-160, 1976. ISSN: 1572-9125.

22. V. Lowenfeld, W. Brittain. *Creative and mental growth*, seventh edition, Macmillian Publishing Co. Inc, 1982.

23. T. Mitsa. *Temporal Data Mining*. CRC Press, 2010.

24. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michael, B. Thirion, O. Grisel, …, E. Duchesnay. *Scikit-learn: Machine Learning in Python*. The Journal of Machine Learning Research, v.12, pp.2825-2830, 2011. Software available at: https://scikit-learn.org/stable/about.html [accessed: 2019-03-27].

25. Python Software Foundation. *Python Language Reference, version 3.6.* Software available at http://www.python.org [accessed: 2018-02-28].

26. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh. *Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping*. Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.262-270, 2012 China.

27. E. Reingold. *Artificial Neural Networks*. Available at: http://www.psych.utoronto.ca/users/reingold/courses/ai/ [accessed: 2019-05-20].
28. S. Salvador, P. Chan. *FastDTW: Toward accurate dynamic time warping in linear time and space*. Intelligent Data Analysis, v.11 n.5 pp.561-580, 2007.
29. R. Schneider, T. Tuytelaars. *Sketch classification and classification-driven analysis using Fisher vectors*. ACM Transactions on Graphics, v.33 n.6, 2014.
30. M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, E. Keogh. *Generalizing Dynamic Time Warping to the MultiDimensional Case Requires an Adaptive Approach*. Data Mining and Knowledge Discovery, v.31 n.1, pp.1-31, 2017.
31. I. E. Sutherland. *Sketch pad a man-machine graphical communication system*. Proceedings of the SHARE design automation workshop, pp.6329-6346, 1964, USA.
32. K. Tanida. 2017. *slaypni/fastdtw: fastdtw-0.3.2*. Software available at: https://pypi.org/project/fastdtw/ [accessed: 2019-03-27].
33. Z. Wen, J. Shi, B. He, Q. Li, J. Chen. 2018. *ThunderSVM: A Fast SVM Library on GPUs and CPUs*. The Journal of Machine Learning Research, v.19 n.1, pp797-801, 2018.
34. D. Widlöcher. *L'interprétation des dessins d'enfants.* Liège: Mardaga, 1998.
35. H. Zhang, S. Liu, C. Zhang, W. Ren, R. Wang, X. Cao.2016. *SketchNet: Sketch Classification with Web Images.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.1105-1113, 2016, USA. Electronic ISBN: 978-1-4673-8851-1.