Facial Animation Retargeting using Recurrent Neural Networks

TEUS VAN OOSTEROM*, Utrecht University, The Netherlands
PROJECT SUPERVISOR: DR. ZERRIN YUMAK, Utrecht University, The Netherlands
SECOND SUPERVISOR: PROF. DR. REMCO VELTKAMP, Utrecht University, The Netherlands

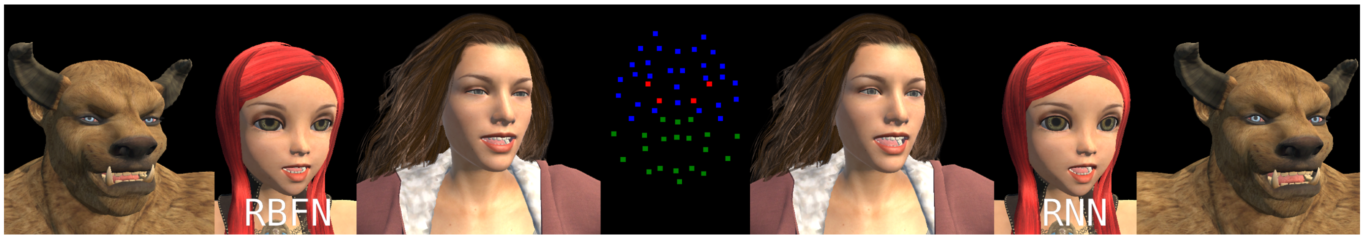


Fig. 1. These screenshots are the visual result of frame 400 of the angry testing set. The left side of the figure shows the RBFN results. The right side shows the RNN results. The motion capture used is the middle images. It is colour-coded, red are the eyes, green is the lower face (tip of the nose and below), blue is the upper face.

ABSTRACT

Games and movies use fully animated characters more and more. Animating these characters manually is a lot of work, and subtle expressions can be missed. Accurate automated mapping of the actor's face to a virtual character is essential to have. To be able to do this, the mapping between an actor's face and the corresponding virtual character needs to be calculated. In a previous paper, an artificial neural network was used, which led to the suggestion to try deep neural networks. No paper has been published using deep neural networks for this problem yet. The usage of a convolutional neural network (CNN), deep belief network (DBN), and recurrent neural network (RNN) is discussed to conclude that RNN shows the most promise of the three. The reason behind this is that RNN can use the prior and future frames to predict the current, which in theory would be helpful for this problem. A radial basis function network or RBFN with Hardy multi-quadric kernel is used as a comparison against RNN. Figure 1 shows a comparison for all characters used for a frame in the angry test set. In that figure, RBFN has the eyes and mouth more closed than RNN has. To determine how RNN compares to RBF a survey was held, and a cost function was made for machine learning results. The survey results show that RBFN is significantly better for three out of the five expressions, all the characters, and in general. The machine learning results show that RBFN performs better as well, with the most significant difference in the mouth area. However, only part of the possibilities of RNN is explored. Various options to improve the results of RNN are listed in the future work section. It is likely that with future research RNN can be improved to be equal or better than RBFN.

CCS Concepts: • **Computing methodologies** → **Animation**; *Machine learning*;

Additional Key Words and Phrases: Facial Animation Retargeting, Deep Neural Networks, Recurrent Neural Networks

*3897095

Authors' addresses: Teus van Oosterom, Utrecht University, Princetonlaan 6, Utrecht, Utrecht, 3584 CB, The Netherlands, teusvanoosterom@hotmail.com; Project supervisor: dr. Zerrin Yumak, Utrecht University, The Netherlands, z.yumak@uu.nl; Second supervisor: prof. dr. Remco Veltkamp, Utrecht University, The Netherlands, r.c.veltkamp@uu.nl.

1 INTRODUCTION

Facial expressions are an essential part of communication. When using just audio or text, people can misinterpret the meaning of words. When using digital characters in movies and games, it is essential that the facial expression delivers that extra bit of information. Animators can give virtual characters facial expressions manually. However, this task is time-consuming, and the animator will probably miss subtle expressions.

Using facial motion capture to capture expressions from an actor and transforming them into a digital character solves these issues. For this transformation, the motion capture data mapping to the blendshape weights is learned. The conventional approach is using RBFN or radial basis function network. RBFN can map non-linear functions linearly by using a kernel function and can be used with a variety of input and output formats. However, RBFN has its disadvantages as well. Two observations were done after implementing RBFN. Looking purely at the eyes, RBFN can result in a blendshape weight that is too low, which leads to the character eyes not closing entirely. For a character with cartoon eyes, it is more noticeable if the eyes do not close properly. Besides this, it looks like the eyes are overreacting to changes of the motion capture data. The overreacting leads to frequent movement of the eyelids, which are not present in the video of the actor. It can also lead to overfitting artefacts [46]. Furthermore, in theory, the representation can be outdone [14]. A proposed approach which on paper should be outperforming RBFN is an artificial neural network or ANN. However, ANN does not result in better representation [14]. It did, however, lead to the suggestion of using deep learning approaches.

In the future work section of Costigan et al. [14] there is a suggestion to use deep neural networks for this problem. No research has been published using deep neural networks for facial animation retargeting. This research first compares different deep neural networks before exploring the possibilities of the best option, which is using a recurrent neural network. Therefore, the research question of this research is: "Does a recurrent neural network to learn the mapping from an actor's face to a virtual model result in a better result than radial basis functions?". To answer this research question a cost function is made, and a user experiment is done. Researching a deep neural network, in this case, a recurrent neural network, for this problem is a proof of concept type of research. Not all possible parameter settings can be evaluated. However, this research gives insight into what does work. Besides this, future work options which have a good chance of providing improvements are found.

The structure of the paper is as follows. First, related work is presented in section 2. Motivations for the presented research are listed in section 3. Specifics about the dataset and processing the data are in section 4. Subsequently, different facial animation retargeting methods are explained in section 5. Results are shown in section 6. The discussion will be in section 7 and the conclusion will be in section 8.

2 RELATED WORK

Research into virtual faces and facial animation goes back for decades. The origins of facial animation lay in 1972 with the first 3D facial animation created [36]. At the 12th SIGGRAPH in July 1985, the first computer animated film was presented [3]. This film was an 8-minute long and was made by four programmers in about four years. The first full-length feature film produced entirely using the technology of computer animation was Toy Story in 1995 [23]. Another significant milestone was in Lord of the Rings: The Two Towers. A system of sculpted faces was created to cover the range of expressions of the character Gollum [45]. There was not any facial motion capture data on Gollum, so animators had to do it all by hand, which was a lot of work. Gollum was the first computer-generated character to be put in a movie with human actors. Over the last decade, more and more research has been done in this field. The progress made resulted in that using facial motion capture became widely applied in games and animation movies.

In sections 2.1, 2.2, 2.3 and 2.4 different facial animation retargeting techniques are presented, which are cross-mapping, parallel parametrisation, manifold-based techniques and expression regularisation respectively. This catergorisation is based on the related work section of Zell et al. [57]. In section 2.5 three neutral networks, deep belief networks, convolutional neural networks, and recurrent neural networks, are explained, with their advantages and disadvantages.

2.1 Cross-mapping

Cross-mapping uses training examples of corresponding facial expressions of the captured actor and the virtual character to learn the mapping. Different techniques can be used to learn this mapping. From basic mappings like piece-wise linear mapping [10] and locally linear embedding [54] to advanced machine learning methods like RBFN [17]. In Song et al. [46] a hybrid retargeting model is made with kernel canonical correlation analysis or kCCa and RBFN to take away the disadvantage that RBFN can overfit. In Kholgade et al. [28] simplicial basis is used. A simplicial basis maps every input expressions as a combination of three expressions using non-negative barycentric coordinates. Bouaziz and Pauly [6] use shared Gaussian process latent variable models or GPLVM. This method can also use frames without knowing the corresponding blendshape weights to train with, which is an advantage.

An advantage of using cross-mapping is that it can work for any character, even non-human like characters, and any facial rig. However, this method highly depends on good and numerous training examples. All expressions used need to be in the training examples for it to work correctly. If one of the testing expression is too different from the training examples, then it probably leads to an inaccurate result.

2.2 Parallel Parametrisation

If two facial rigs are semantically equivalent, transferring an animation is easy. This process copies the control parameters of one facial rig to another. However, having semantically similar facial rigs is not that easy. Doing this manually costs much time, requires excellent modelling skills and a good knowledge of the anatomy of the face. Several approaches have been proposed to automatically transfer blendshapes from a generic face model to a neutral target.

Noh and Neumann [33] use a source blendshape rig and the targets neutral face. Between the source and target, a dense surface corresponds is established. Then for each expression, the per-vertex displacement is transferred. This approach is not perfect, and manual improvement is required, but this requires less time than doing it manually from the start. Sumner and Popović [47] is based on this approach. It requires a manually created correspondence map between the triangles of the source and target, which requires less manual labour than Noh and Neumann [33]. RBFN in Orvalho et al. [34], Seol et al. [42], and Seol et al. [44] are approaches based on Noh and Neumann [33].

Various improvements on RBFN have been suggested. Li et al. [31] uses a set of example poses of a target character to generate facial blendshape rigs. This set of examples have to be created manually which requires some work. Saito [39] uses contact and smoothness constraints to prevents penetrations or separations. Xu et al. [56] decomposes high-fidelity facial performances into high-level facial feature lines, large-scale facial deformation and fine-scale motion details. The final result is easily editable for the final touches. Bouaziz et al. [7], Ichim et al. [25], and Seol et al. [43] have a different approach, which is to use iterative refinement schemes for real humans. This approach only works for targets with a human-like structure.

Parallel parametrisation approaches all require the target and source model to be semantically equivalent. When this is not the case, these approaches will often fail to preserve the facial expression. This failure can lead to unnatural face deformations if the target needs smaller weights than the source. Seol et al. [42] uses velocities over a sequence of captured frames to prevent artefacts. In Zell et al. [57] the transferred blendshapes automatically adapt to the actor's range of motion. This approach requires a motion sequence of an actor and the sparse correspondences between the source and the target.

2.3 Manifold-based Techniques

The current progress on transfer learning for classification, regression and clustering problems in categorised and reviewed in Pan and Yang [35]. Aligning the ranges of motion between the source and the target in Zell et al. [57] is based on this. Using this, transfer results are improved while preserving the geometric structure.

Fan et al. [19], and Wang and Mahadevan [52] use another transfer learning approach. This approach uses unsupervised learning to be able to use frames without the corresponding blendshape weights as well. A different approach is using automatic translation or image set matching in Cui et al. [15] and Pei et al. [37]. While not directly related to facial animation retargeting, the methods proposed in both papers can be used in the transferring stage. Fan et al. [19], and Wang and Mahadevan [52; 53] solve the eigendecomposition of the graph Laplacian to transfer between embedding spaces. Combining dimensionality reduction can optimise these embedding, which is the crucial aspect of manifold-based techniques, with additional constraints.

The main advantage of using dimensionality reduction is that when reduced to a low enough dimensional space Euclidean distance can be used to describe the similarities between the data sets. When manifold-based techniques are used to transfer blendshapes from source to target, the proportions and ranges of motion of the source should match.

2.4 Expression Regularisation

Different approaches are used to reduce the number of artefacts in blendshape animation. Restricting blendshape weights to a fixed interval is used in Bregler et al. [8], and Chuan and Bregler [12]. In Seo et al. [41] large weights get penalized. However, the downside of using these approaches is that a valid combination of blendshape weights is not possible anymore because of these restrictions Seol et al. [44]. It can also lead to an invalid combination of blendshape weights being allowed.

Principal component analysis or PCA is used to reduce the number of features. PCA-based priors are used in Anjyo et al [1], and Lau et al. [30] for direct blendshape manipulation and in Seol et al. [42] for retargeting. Lau et al. [30] uses PCA to learn a statistical model. This model defines the prior term and creates a constraint that the generated facial expressions are natural. Seol et al. [42] uses PCA to create an eigenvector basis and corresponding eigenvalues. Multivariate normal distribution is done on the PCA data to construct a prior model.

A different approach is using smooth skin deformation. For this, a prior can be used to penalise surface deformations. Comparable approaches are proposed in Barrielle et al. [2], Bicket et al. [4], and Ichem et al. [26].

2.5 Neural Networks

In Costigan et al. [14] ANN is compared to RBFN. The testing errors were similar while ANN is harder to use. In the future work section is a suggestion to use deep learning methods and compare it to RBFN. Three methods are compared to choose which one to use.

CNN or convolutional neural network is a deep, feed-forward artificial neural network. It is used for image and video recognition [29], recommendation systems [49] and natural language processing [13] among other things. An advantage of using CNN is that it needs minimal preprocessing. However, the computation cost is relatively high.

DBN or deep belief network is a deep neural network. The top layers have undirected, symmetric connections and the lower layers have top-down directed connections. It is used for EEGs [32] and drug discovery [21] among other things. DBN was one of the first effective deep learning algorithms when it is trained greedily [24]. An advantage of DBN is that it can do unsupervised learning to become a feature detector. However, the effect of unsupervised learning is not yet known.

RNN or recurrent neural network is an artificial neural network. It is composed of LSTM, long short-term memory, units. An LSTM unit has a cell, forget gate, input gate, and the output gate. It is used for handwriting recognition [22], speech recognition [40] and speech animation [38; 48] among other things. An advantage of is that it uses multiple time steps, so it uses one or more prior frames to predict the current frame. RNN can also use bidirectional LSTM or BLSTM like done in Sadoughi and Busso [38]. Bidirectional uses the future besides the past to predict the current frame. A disadvantage of RNN is that the more time steps used, the longer the computation time is.

3 MOTIVATION

Facial animation retargeting uses the actor's facial motion capture data to manipulate the face of a virtual character. This manipulation is done by calculating a mapping to go from the motion capture data to blendshape weights. Numerous approaches are listed in the related work section, section 2. Of those solutions, RBFN with the Hardu multi-quadric kernel is the current standard [18; 46]. Machine learning technics have been used for this problem. Like Costigan et al. [14], which uses an ANN. The results were similar to RBFN while it requires a lot more work. The future work section of Costigan et al. [14] suggested using a deep neural network. This research is exploring that suggestion. In section 2.5, the advantages and disadvantages of CNN, DBN, and RNN are listed. Of those three RNN has the most potential in theory. The contributions of this research are as follows:

- No papers could be found using a deep neural network for facial animation retargeting. The results of using RNN do not indicate if another deep neural network could be better than RBFN. However, the shortcomings of RNN can be used to theorise if another deep neural network could be better than RNN.
- Since no paper using a deep neural network could be found, a paper using a recurrent neural network for facial animation retargeting has not been found. This research is just a start of researching if RNN could be better than RBFN. Many parameter settings could not be evaluated in this research due to time constraints. This research shows the parameters and range of those parameters explored, which combination is the best. A lot of future work options are found which could improve the results from RNN. These options are listed in section 7.2.

4 DATA

A facial motion-captured dataset needs to be picked for this research. To choose a dataset a comparison paper [55] was used. In table 1 of that paper, fourteen datasets are listed. A requirement of the dataset for this research is that the person getting recorded needs to speak. The person needs to talk English. Also, a single person needs to have plenty of data. Using different people in one training set is a complect problem which is not explored in this paper. Also, the dataset has to contain different expressions. Selecting a dataset which is equally divided over all the expressions can quickly be done then. Multiple datasets have these requirements. The IEMOCAP dataset [11] was picked, since it has over an hour of data at 120 fps for each actor on average. This dataset should contain enough data for RNN. Another advantage of the IEMOCAP dataset is that the videos are manually evaluated to list the timestamps for each expression in all the videos.

The IEMOCAP dataset uses Facial action parameters or FAPs. This dataset has 53 facial markers, as shown in 22 in appendix C. The RHD and LHD are used to determine the rotation and position of the head. It does not influence the face. The hand coordinates are not used. The expressions listed in the manual evaluation are limited to angry, frustrated, happy, neutral, sad and other. The dataset has five sessions. Each session has a male and a female talking to each other. Each recording has one person being motion captured, so each session happens twice. The data of the female from the first session is used throughout this research. In theory, the optimised parameters should also give good results using another person its motion capture data. However, this has not been tested.

The dataset has some not a number values, and some values which are not humanly possible. Therefore the dataset was first preprocessed. This is explained in section 4.1. The results of RNN have a lot of plateaus and sudden jumps. To smooth this out postprocessing is used. This is explained in section 4.2 respectively.

4.1 Preprocessing

Some markers could not be captured resulting in not a number or NaN values in the dataset. To fix this issue for a NaN value linear interpolation is used between the previously known value and the next known value to replace the NaN value. Besides this problem, some positions of the markers are not possible. For example, having an eyelid marker lower than the RC8 or LC8 marker from figure 22 in appendix C. It is not possible that a marker moves a lot in a single time step, so for that reason if a value changes more than 0.5 in a time step it is altered similarly as the NaN problem. The value of 0.5 is defined as a maximum of normal movement by looking at the dataset and the resulting picture of the motion capture values. Here it takes the previous value and finds the next value which differs less than 0.5 times the time steps taken from the last value to this value. This approach might not fix consistent outliers. If there are enough frames with similar outliers, this approach will smooth out the transitions towards the outliers and back to a normal range.

4.2 Post-processing

The changes of the test results of RNN are jumping up and down between values. The current frame may have the minimum value and the next frame the maximal value. The results are the same for a couple of frames, and then it changes a lot in a single frame. The significant changes resulted in videos where the face was changing abruptly. For example, the right eye in the angry results has for frame 191 the value 0.18 and for frame 192 the value 0.52. This sudden jump in blendshape weight results in non-human-like behaviour. This issue is solved by post-processing. Given the current frame, it looks to the tenth frame ahead. The current frame new value is the old value plus the difference of the previous frame and the tenth frame ahead divided by 11. This value is capped at ±0.05 to smooth out the outliers. The result loses some predictions of the model, but the video shows smooth transitions, which is more important.

5 APPROACH

This research compares RNN with RBFN. Fourteen blendshapes are used for both approaches. These are: angry, frown, smile, surprised, mouth open, mouth narrow, right eye squint, left eye squint, right eye closed, left eye closed, right brow up, left brow up, mouth smile, and mouth frown. Throughout this section, the cost is used to determine which option is better. To calculate this, predicted blendshape weights are translated back to motion capture values. The Euclidean distances between these values and the actual values are used to determine the cost.

In section 5.1, the method which calculates the blendshapes weights of the training data for both RBFN and RNN is explained. Then in section 5.2, RBFN is explained. Five training set selection approaches are evaluated to determine which one to use as a comparison. At last in section 5.3 RNN is explained. RNN has many parameters. In section 5.3.1 reducing the number of possible blendshape weights predicted by a single model is tested. In sections 5.3.2, and 5.3.3, the number of hidden layers and the reduction of the dimensions of the input are evaluated. Then in section 5.3.4 increasing the timesteps is tested. Using the future frames besides the previous frames using a bidirectional LSTM or BLSTM is evaluated with and without regions. In sections 5.3.5, 5.3.6, and 5.3.7 using a dropout layer, decreasing the batch size and increasing the iterations are tested. Then in section 5.3.8 using multiple LSTM cells is evaluated. The last section, section 5.3.9, combining multiple LSTM cells in combination with using more iterations is tested. Besides this, the best setup found is listed.

5.1 Motion Capture Driven

Both RBFN and deep neural networks need to have motion capture data with the corresponding blendshape weights. Determining those weights manually is a labour intensive job. The following formula, inspired from Joshi et al. [27], estimates the blendshape weights automatically by minimising the difference between the motion capture data and the data created by the blendshape weights:

$$\min \sum_{j=1}^{k} (m_j - (neutral_j - (\sum_{i=1}^{n} \alpha_i V_{ij})))^2$$

Where m is a single motion capture row and k is the amount of markers times three, since each marker has xyz coordinates. *neutral* is a motion capture data row corresponding to a virtual face with all the blendshapes weights at 0. In the formula, n is the

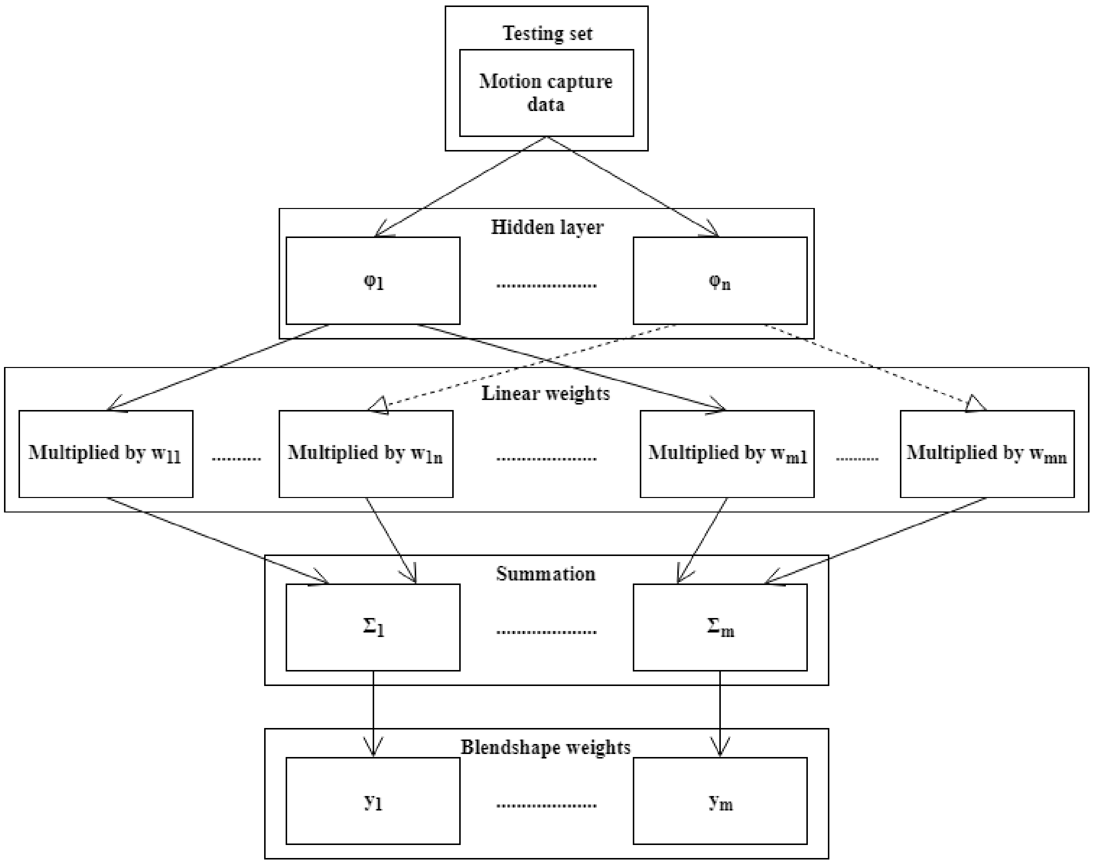


Fig. 2. The pipeline used for RBFN. The number of hidden layers is the same as the number of training examples. The first digit of the weight is the blendshape, the second digit is the hidden layer. The setup used in this paper has 100 hidden layers and 14 summations and blendshape weights. Two types of arrows are used for visualisation reasons only.

number of blendshapes, and α_i the blendshapes weight. The formula calculates the blendshapes weights of blendshape i. The range of the weight value is between 0 and 1. V_ij is the displacement of point p_j compared to $neutral_j$ when blendshape i its weight is set at 1. For each motion capture row, this formula is used separately, since they do not influence one another. This formula minimises with a best improvement hill climbing algorithm. Due to the nature of the problem, it is not likely that the optimum found is only a local optimum since blendshapes do not contradict each other. This formula is used to create training examples for RBFN, which is explained in section 5.2, and for RNN, which is explained in section 5.3. The frames which need corresponding blendshapes is then the input, and the output is those blendshape weights. The cost calculation is the reverse of this function. Using this function only will have a low-cost function and can be lowest possible. It is the lowest if the results of the hill climber for each frame are global optima. However, in the dataset even after pre-processing, there is still noise. If a video is made with these results, it will sometimes show a shaking mouth, eyelids, and brows. There is also a synchronisation issue with the eyes, where the left eye is halfway closed while the right eye is fully open. This behaviour is unnatural. Therefore, the cost function can not only be used to compare RBFN and RNN. A survey with video results needs to be used.

This formula does differ a bit from Joshi et al. [27] since they do not use $neutral_j$. It is used there in combination with RBFN. The formula presented above needs to have this term, without it, the eyes never open. For example, the y coordinate of the left eye closed is closer to the displacement then open. Without using the neutral, the function will always return a blendshape weight of 1. Using *neutral* prevents this issue.

5.2 Radial Basis Function Network

Radial basis function network or RBFN was first proposed in Broomhead and Lowe [9]. Originally RBFN was invented for military usage. However, it is used in a lot more fields including but not limited to facial animation retargeting. The following formula is used to calculate a single blendshape weight:

$$f(x_i) = \sum_{n=1}^{N} w_n \phi_n(x_i)$$

Where x_i is the input motion capture row and $\phi_n(x_i)$ is the kernel function. The training data is used to learn w_n is a linear weight.

This linear weight is not the same as a blendshape weight. Each blendshape has its own linear weights. N is the amount of training data used to get the weight vector. The testing process is shown in figure 2. The input is the testing set. Each hidden layer calculates its kernel. The number of hidden layers is the same as the number of training examples. This result is multiplied by the linear weights, which are different for each blendshape. The linear weights are calculated in the training phase. For each blendshape, a summation of the results of the multiplication is done. The outcome is a single weight for each blendshape.

The kernel function used is the Hardy multi-quadric from Song et al. [46]. For facial animation retargeting, RBFN with the Hardy multi-quadric kernel is the current standard [18; 46]. Therefore, comparing the results of RNN with this combination is the best way to see how good RNN is. The following formula is the hardy multi-quadric kernel:

$$\phi_j(x_i) = \sqrt{|x_i - x_j|^2 + s_j^2}$$

Where the square root of the distance between the x_i and x_j plus s_j term, which is the distance between x_j and the closest point. This can be calculated in the following formula:

$$s_j = \min_{j \neq i}(x_j - x_i)$$

To be able to calculate blendshape weights the linear weights vector $\boldsymbol{w}$ needs to be calculated first. For this, the blendshape weights calculated in the previous section is used. Let $\boldsymbol{t}$ be a vector of the blendshape weights of one blendshape and $\boldsymbol{H}$ be a matrix with $H_{ij} = \phi_j(x_i)$. The previous section, section 5.1, explains how $\boldsymbol{t}$ is calculated. The following formula shows the relation between the variables:

$$\boldsymbol{t} = \boldsymbol{H}\boldsymbol{w}$$

Since $\boldsymbol{w}$ is the variable that needs to be learned, the formula needs to be rewritten. Taking the inverse of $\boldsymbol{H}$ solves this issue. Then the following formula can be used to calculate $\boldsymbol{w}$:

$$\boldsymbol{H}^{-1}\boldsymbol{t} = \boldsymbol{w}$$

Having determined the value of $\boldsymbol{w}$ this RBFN can be used to calculate the value of the blendshape weight for input x_i.

Different papers use a different amount of training data for RBFN. Dutreve et al. [18] uses a training set of size 25. However, in that paper, only 24 motion capture markers are used. Costigan et al. [14] uses a training set of size 33. The dataset of that paper consists of 59 motion capture markers. If a training set is too small compared to the dimension of the tracking data, the model might underfit [5]. For that reason, Costigan et al. [14] uses PCA with a variance of 85% and three regions to reduce 177-dimensional tracking data to 16 principal components. The three regions used are the lower face, upper face, and eyes.

Using all the possible training data is not an option. The reason is that both the training and testing phase will take too much time. Inversing the $\boldsymbol{H}$ matrix is the biggest issue for the training phase. The big O notation of inverting a matrix is $O(n^{2.373})$. If all the possible training data is used, the training phase will probably take months. The testing phase will also be slow since for each training example the kernel needs to be calculated. Therefore, two main approaches were evaluated, picking at random equally divided over the five expressions and picking the min and max values. Picking the min and max values was divided in picking for each of the coordinates of each marker and picking for each marker. A formula calculates the distance from the marker to the origin to find the min and max. If a row is multiple times a min or max, it will be selected only once. Picking at random was done for 100 to 300 rows in steps of 100. The results can be seen in figure 3. The values used to create this figure can be found in table 6 in appendix B. The figure shows that picking 100, 200, and 300 training frames at random result in about the same result. The data in the table shows that picking 300 training frames at random is the best. However, using more training data results in a slow training and testing phase. The training phase only happens once, so that can be neglected, but having a slow testing phase is not desirable. Besides this, Dutreve et al. [18] and Costigan et al. [14] do not use that many training frames. Therefore, picking 100 training frames at random was chosen to have it run fast, but still, have good results.

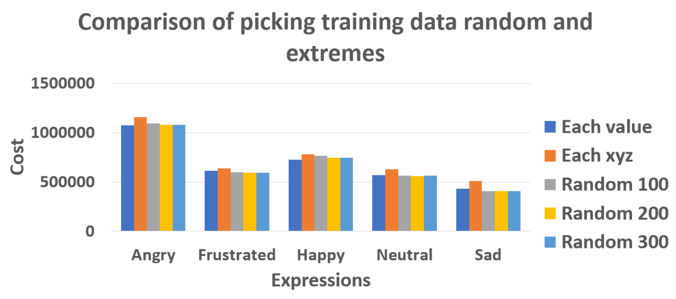


Fig. 3. Picking the dataset random or picking the min and max of each value are about the same in the results. However, the training and testing time increases rapidly. Therefore, the best option is random 100, since it is a good balance between training and testing time and the result.

5.3 Recurrent Neural Network

A recurrent neural network or RNN uses motion capture data of multiple frames to predict one frame. For an LSTM usually the current and previous frames are used. For a BLSTM the future frames can be used as well. This is explained in section 5.3.4. The training set used has sequences of motion capture data of the same length. In this section, two training sets are used. One for the sections 5.3.1, 5.3.2, and 5.3.3 and one for the sections 5.3.4, 5.3.5, 5.3.6, 5.3.7, 5.3.8, and 5.3.9. The first training set has the corresponding blendshape weights of the last frame of each sequence since it uses only the previous frames. The second training set has the blendshape weights of the middle frame of each sequence so that each sequence has the same amount of previous and future frames. To test this approach and to test if multiple previous and future frames help a new test set was drawn for the section 5.3.4 and beyond. The calculation of these blendshape weights is explained in section 5.1. The training set used for the first three experiments only has a time step value of 2. Both training sets consist of 100000 training examples, equally divided

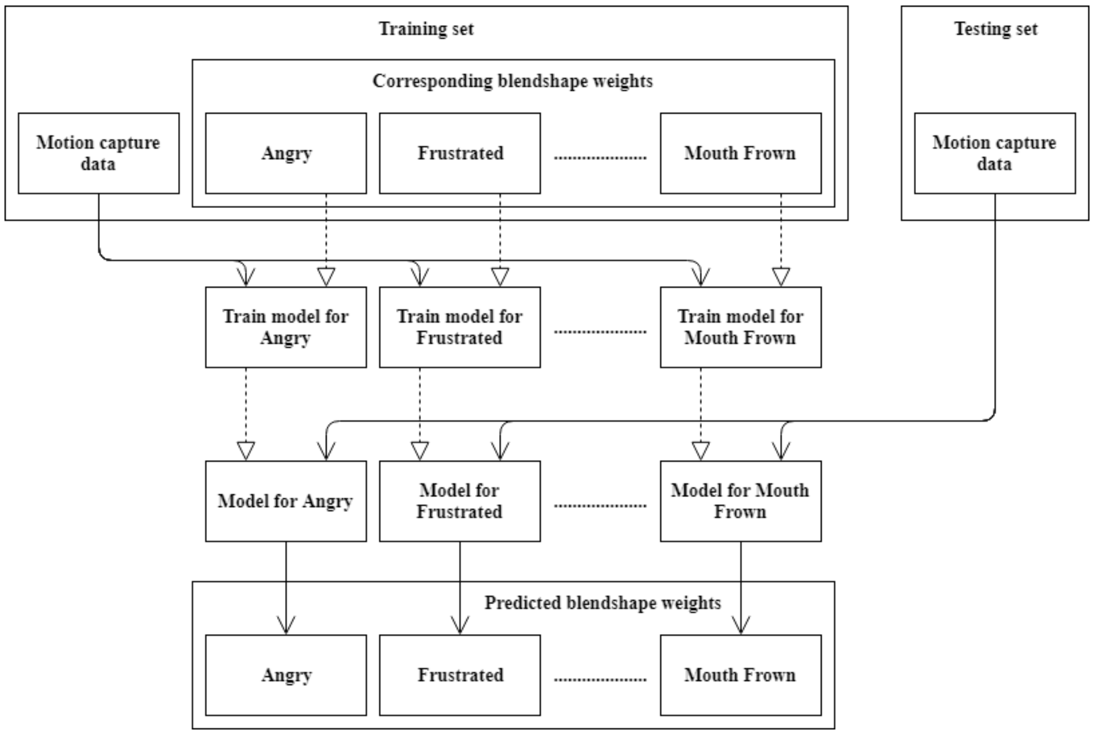


Fig. 4. The pipeline of predictions. Two types of arrows are used for visualisation reasons only. This image illustrates that each of the fourteen blendshape weight, which are listed in 5.

over the angry, frustrated, happy, neutral, and sad expression. Each blendshape has its own model because the RNN implementation of tensorflow expects one value as the weight. Multiple models are used for each blendshape, the reason and how it works is explained in section 5.3.1. The output of each model is an array of odds for each possible predictions. The highest odd is used as the prediction. For BLSTM this is slightly different, which is explained in 5.3.4. At last, the predictions for each blendshape are combined in an array and written away to a text file. This process is shown in figure 4.

In this section, five-fold cross-validation is used for all the experiments. Cross-validation is used to optimise the parameters in general, not just for the testing set. The folds are picked random, but using the same training set for different experiments always results in the same folds being used. To be able to find the optimal parameters, all tests in this section are run on the CPU because running it on the GPU is not deterministic. Running on the GPU could lead to parameter settings suddenly performing good or bad, while if it is repeated it could yield the opposite result. Running tests on CPU does, however, run a lot slower than on the GPU.

First splitting up one prediction for all the possible weights is evaluated. This is done in section 5.3.1. Then in section 5.3.2 the hidden layer parameter is evaluated for the values 400 up to and including 800 in steps of 100. Reducing the dimensions of the motion capture data input is explored in section 5.3.3. Reducing the dimensions is done by defining region-specific and blendshape-specific input. Besides the previous frames, the future frames can be used as well. Using the future frame and time step values, which is the number of frames backwards and forwards, from 2 up to and including 6 are evaluated in section 5.3.4. A dropout layer randomly drops input data each in each training iterations. Using a dropout layer prevents overfitting. The results of using a dropout chance of 25% and 50% are shown in section 5.3.5. The batch size parameter is set to 200 for all these experiments. Lowering this parameter to 20 and 100 is tested in section 5.3.6. In section 5.3.7 the iterations for the first prediction is increased to 100000 and evaluated. The number of basic LSTM cells used is explored in section 5.3.8. The number of basic LSTM cells is tested for up to four cells. The last section, section 5.3.9 combining the results from sections 5.3.7 and 5.3.8 is evaluated. In this section, the final parameter values are listed as well.

5.3.1 Multiple predictions. Using one model to predict all possible 101 weights, 0 up to and including 100, 80% of the predictions is 0. This is way too much 0 predictions, which results in a high cost. The reason why there are so many 0 predictions might be because the training set has 0 as a blendshape weight 39% of the time. Other blendshape weights do not have the same amount of representation. The differences in the amount of representation for each possible blendshape weight is most likely the reason for this. Therefore, splitting the prediction up into multiple smaller predictions might result in an improvement. For this, three methods of dividing the prediction up are explored. These are ceil, floor or round the number to dozens for the first prediction. These methods reduce the number of possible predictions to 11. Almost all possible predictions get a

Regions	Markers
Eyes	RC3, RC7, RC8, RLID, LC3, LC7, LC8 and LLID
Mouth	CH1, CH2, CH3, MOU1, MOU5, MOU6, MOU7 and MOU8
Brow	RBM0, RBM3, RBRO1, RBRO2, RBRO3, RBRO4, LBM0, LBM3, LBRO1, LBRO2, LBRO3 and LBRO4

Table 1. Markers used for specific regions.

higher amount of representation, which is the major advantage. This can prevent the issue of too much 0 predictions. However, this could prevent an optimal prediction. If the training data of the optimal blendshape weight is weakened by the motion capture data it is combined with, another of the 11 possible blendshape weights can be predicted. The first and second prediction combined is the final result then. The reason to use any of those function to reduce it to 11 possible predictions is to balance the number of possible predictions for the first prediction and the second prediction. Using any of those methods will result in second predictions with 10 possible predictions, except for the first and last possible prediction for the first prediction using the round function. Those have 5 and 6 possible predictions respectively. The results can be seen in figure 5. The values used to create this figure can be found in table 7 in appendix B. These results are found using 10000 iterations for the first prediction with a batch size of 200, a time step value of 2, 400 hidden layers and a learning rate of 0.001. For the second prediction 1000 iterations and a batch size of 100 is used. The figure shows predicting all possible blendshape weights in one model, which is basic, is a lot worst compared the other options, so having multiple smaller predictions does help. From the three options for splitting the data, using ceil is the best option. Table 5 in appendix B shows the prediction of the blendshape weights 0 up to and including 9. This range is specifically looked at since 0 does have the biggest representation, so the biggest impact is expected in this range. The table shows that after basic, ceiling has the most 0 predictions. The round function combines blendshapes 0 up to and including 4 together for the first prediction. This combination has more data than just blendshape weight 0 which is used with the ceil function. However, the first prediction results in a lower total than the prediction for 0 using ceil. The floor function combines blendshape weights 0 up to and including 9. This combination has even more data. Using the floor function results in a higher total than the prediction for only blendshape weight 0 with the ceil function, however, the second prediction spreads it out resulting in a lower number of 0 predictions than the round function. The reason the ceil function is the best is that other data do not weaken the data of blendshape weight 0, and other blendshape weights combined have enough representation to get more predictions.

These findings are all done with selecting random training examples equal divided over the five expressions. However, the issue with all possible blendshape weights in one model not having equal representation could be solved differently. A possible option would be to calculate the blendshape weights of all the possible training data and pick each blendshape weight the same amount of times. This idea is listed in the future work section, section 7.2.

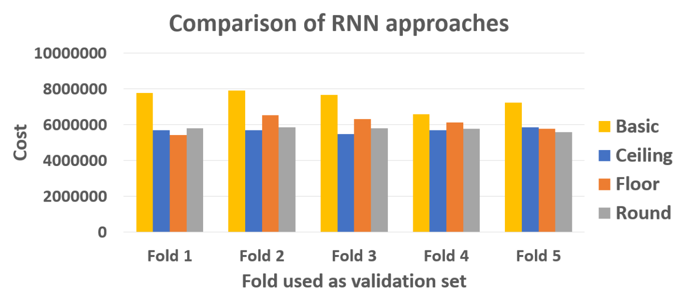


Fig. 5. Predicting all possible blendshape weights in one model, which in this picture is basic, is not a good option. From the three options to split it up in smaller predictions, ceiling advanced is the best one.

5.3.2 *Hidden layers.*

The number of hidden layers is representing the learning capability of the model. Increasing this value could learn valuable features. However, it could also learn bad features which are counterproductive. Hidden layers are used to be able to express a non-linear function, which helps because the mapping of the actor's face to a virtual character is non-linear. This evaluation is done for 400 to 800 hidden layers with steps of 100. The rest of the setup is the same as used for ceiling in section 5.3.1. The results are shown in figure 6. The values used to create this figure can be found in table 8 in appendix B. It shows that using 700 hidden layers is optimal. However, as shown in figure results from each fold vary a lot. There is not a clear trend visible in the figure, so there could be a different number of hidden layers which results in a better result. However, looking at different parameters was prioritised since there is not much difference between using 500, 600, 700, or 800 hidden layers.

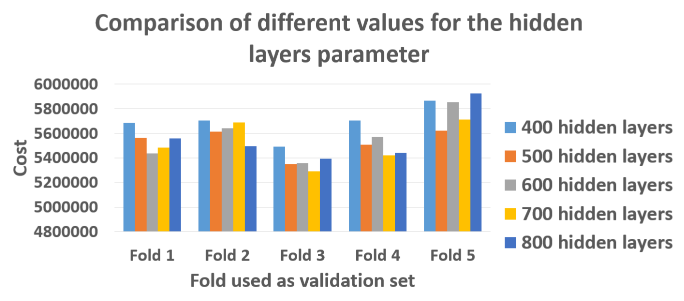


Fig. 6. The best option, determined by the total sum, is 700. For fold 3 and 4 it is the best option, for fold 1 and 5 the second best and for fold 2 the second worst. This shows that there is not a clear pattern.

Blendshape	Markers
Left eye closed	LLID
Right eye closed	RLID
Left eye squint	LC3, LC7 and LC8
Right eye squint	RC3, RC7 and RC8
Mouth open	CH1, CH2, CH3, MOU1, MOU5, MOU6, MOU7 and MOU8
Mouth narrow	X coordinate of MOU1 and MOU5
Mouth smile and mouth frown	MOU1, MOU2, MOU4, MOU5, MOU6 and MOU8
Left brow up	LBM0, LBM3, LBRO1, LBRO2, LBRO3 and LBRO4
Right brow up	RBM0, RBM3, RBRO1, RBRO2, RBRO3 and RBRO4

Table 2. Markers used for specific blendshapes.

5.3.3 Reducing the dimensions of the input. 53 facial markers were used to create the dataset, as explained in section 4. The dimensions of a frame,159, is multiplied by the time step value. An idea from Sadoughi and Busso [38] is to use part of the data from some of the blendshapes. For example, the mouth does not influence the eyes, so the model should not suddenly find such a link. Besides this advantage, both the training and testing phase will be quicker. An evaluation is done for two approaches. The first one is using regions. Regions divide the blendshapes into four regions, whole face, mouth, eyes, and brows. The blendshapes angry, frustrated, smile and surprised are part of the whole face. For this region, nothing changes. Eyes closed and squint both left and right are part of the eyes region. Mouth open, mouth narrow, mouth smile, and mouth frown are part of the mouth region. The last region is brows which contain left brow up and right brow up. The markers used for each category are listed in table 1. In figure 22 in appendix C these marker codes are explained. Using these regions prevent markers which have nothing to do with a blendshape to influence the outcome of the prediction. The second approach is to use blendshape specific markers. This approach reduces the number of markers even further. For each blendshape, the markers used are listed in table 2.

The parameters are the same as in section 5.3.2 except for the parameter hidden layers, which is set at 500. The results of this experiment are shown in 7. The values used to create this figure can be found in table 9 in appendix B. Overall regions is the best option. Blendshape-specific might not work because small features which influence the prediction are removed.

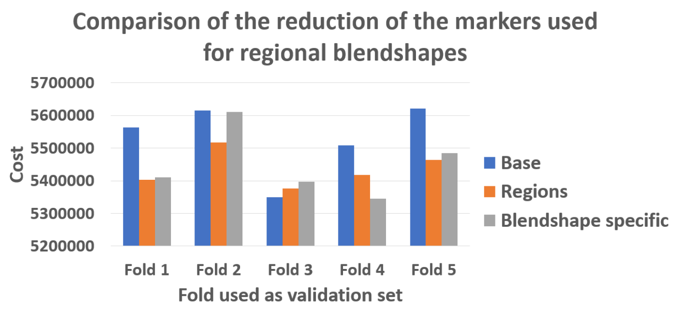


Fig. 7. The best option, determined by the sum, is regions. While it can be outscored on specific folds, the cost is relatively stable.

5.3.4 BLSTM and time steps. The previous tests were all done with a time step of 2. The time step value indicates how many frames are used. A time step value of 1 will only use the current frame. To test if different values for this parameter will improve the results a new training sample was drawn picking 11 consecutive frames. The blendshape weights of the middle frame of those 11 are calculated. This calculation is explained in section 5.1. An experiment was done changing the time step value up to 6 with steps of 1. An advantage of having a higher time step is that it can find a trend between time steps. The computation time does increase, however, as explained in section 5.3.3. In figure 9 the results of the experiment are shown. The values used to create this figure can be found in table 10 in appendix B. The rest of the parameters used in this experiment are the same as section 5.3.1. Using a time step value of 3 is the best option. However, no clear pattern is visible in the results.

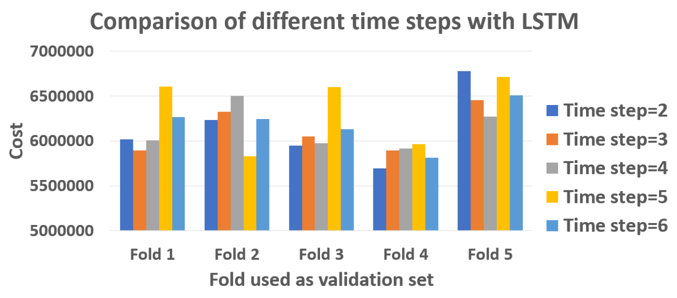


Fig. 9. A time step of three is the best of these five. However, a time step of six is better than five, so this might be a local optimum.

This approach has the preceding frames predict the current frame, while the future frames can help the prediction. Bidirectional LSTM uses the preceding and future frames. This process using BLSTM with a time step value 3 is shown in figure 8. Two frames backwards are used with the current frame in the backwards dataset, and two frames forwards are used with the current frame in the forwards dataset. The forwards and backwards data both predict the first prediction. Those predictions are multiplied. The idea of multiplying them comes from the f-score [51]. The first prediction is used for a second prediction for both the forwards and backwards data. Again these predictions are multiplied. With the result of the multiplication, the end prediction is made. The results can be found in figure 10. The values used to create this figure can be found in table 11 in

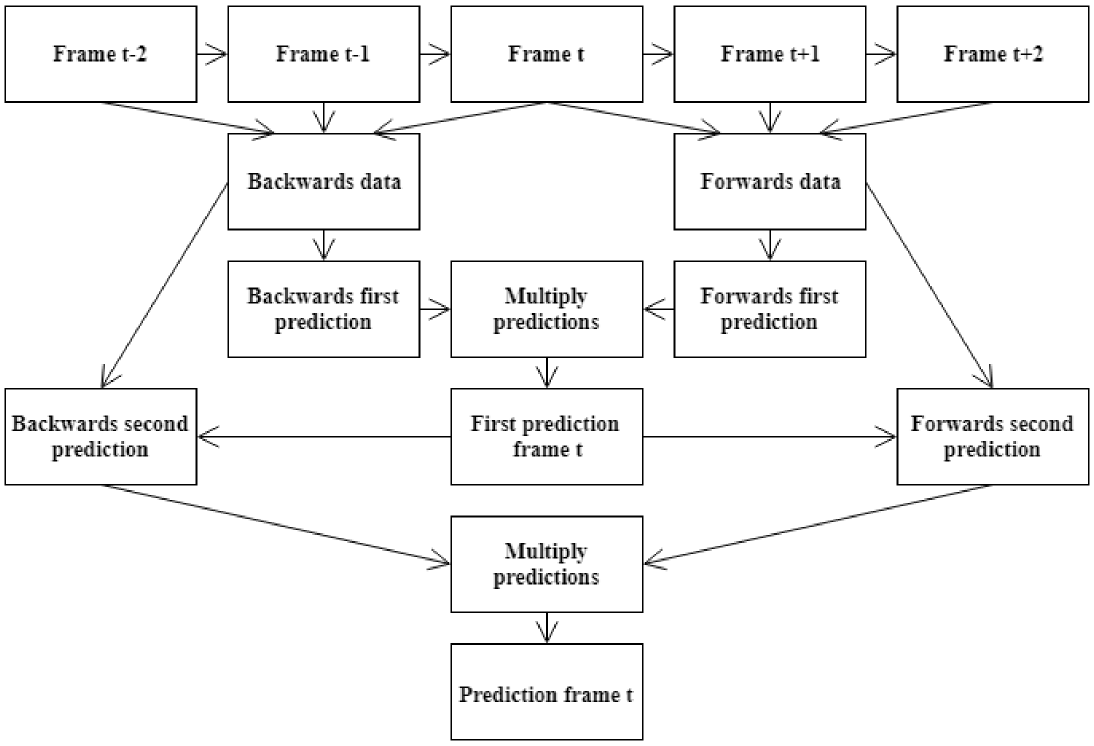


Fig. 8. The pipeline used for BLSTM with time step value 3. LSTM has the backwards prediction as frame t prediction.

appendix B. The rest of the parameters used in this experiment are the same as section 5.3.1. This figure shows the same pattern as in table 10 with the time step value of 3 being the best, but no clear pattern is present in the data. The results show that BLSTM outperforms LSTM.

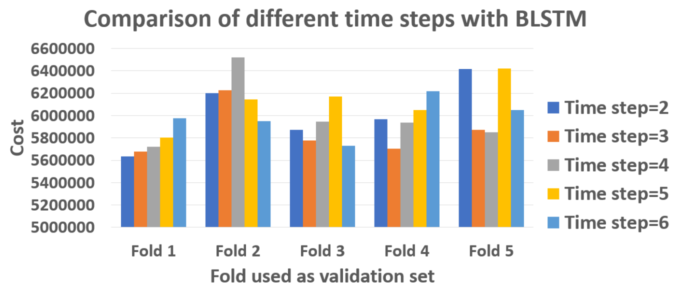


Fig. 10. Similar to figure 9 a time step of three is the best option. However, a time step of six is better than a time step of five, which leaves the question again if a time step of three is a local or global optimum.

Both regions and BLSTM have proven that separately the result in improvements. However, combining them also needs to be tested. The result of using both together is shown in figure 11. The values used to create this figure can be found in table 12 in appendix B. The rest of the parameters used in this experiment are the same as section 5.3.1. Unlike the previous two tests, the time step value 2 is the best. However, like BLSTM and LSTM without using regions, time step value 6 is better than time step value 5. Exploring if increasing the time steps result in a better result is listed as future work in section 7.2. It was not explored in this research because a new dataset had to be drawn with more consecutive frames for each training sample. Due to this, time step value 2 up to and including 6 need to be reevaluated as well for that dataset. Timewise this would prevent this research looking into other parameters as well.

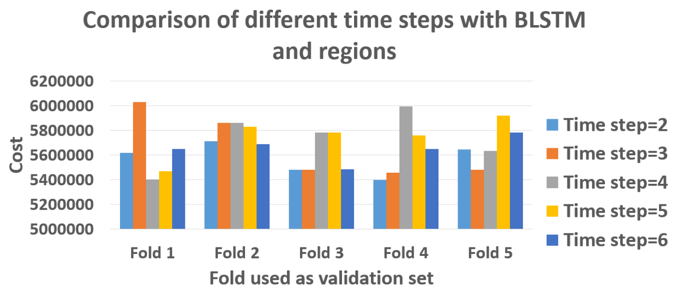


Fig. 11. A time step of two is the best option. However, like figure 9 and figure 10 there is not a pattern which proves that increasing the time step further will not yield a better result.

5.3.5 Dropout layer. To improve this results different adjustments have been explored, such as adding a dropout layer. A dropout layer randomly drops input training data from each training batch used, which helps prevent overfitting. Multiple papers [38; 48] use a dropout layer. A dropout chance of 0.2 and 0.5 is used in those papers respectively. However, with a dropout layer with the chance of 0.25, a single fold had the value 15190723, 0155825, which is a little less than three times the best result. A dropout layer with a chance of 0.5, the result of a single fold was 10451954, 0207968, which is a little less than two times the best results. Both tests use the same parameters as used for BLSTM with regions in section 5.3.4. Using a dropout layer does not work according to these results, therefore running the rest of the fold is a waste of time. The model might simply not have been trained enough for a positive effect using a dropout layer. Increasing the iterations purely for the dropout layer to have a positive effect would not be a good idea. Because then the model is overfitted to have a dropout layer work, instead of using a dropout layer when the model is overfitted.

5.3.6 Batch size. Another adjustment was to use smaller batch sizes. Each iteration only learns from a batch with the size of the batch size parameter. A smaller batch size might result in better features within the motion capture data being learned. The total values over all five folds is shown in figure 12. The values used to create this figure can be found in table 13 in appendix B. These experiments are done with the same parameters as used for BLSTM with regions in section 5.3.4. Lowering the batch size resulted in worse results. The lower batch size does result in less data being used overall since the number of iterations is still the same. Countering this effect by increasing the number of iterations so that the iterations times batch size is the same could yield interesting results. Then the effect of using the same amount of data but in smaller batches can be seen. This idea is listed as a future work option in section 7.2.

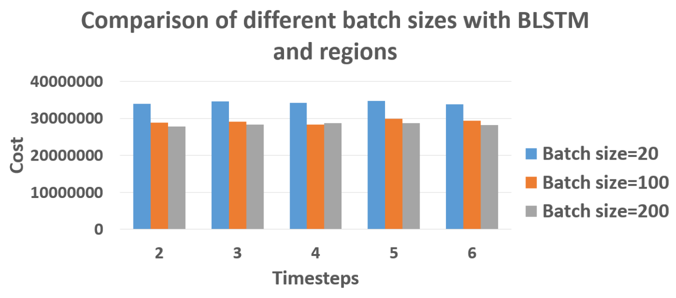


Fig. 12. Decreasing the batch size leads to a worse result. This is probably due to underfitting it a bit at 100 and severely underfitting it at 20.

5.3.7 Increasing the iterations. The model is trained with different batched for the number of iterations used. The iterations for the first prediction were set at 10000 at the start not to be low enough to prevent underfitting, but also not be too high which results in longer computational times. Increasing the iterations for the first prediction to 100000 was explored. These results are shown in figure 13. The values used to create this figure can be found in table 14 in appendix B. The same parameters as used for BLSTM with regions in section 5.3.4 are used. It shows that in four out of the five cases 100000 iterations is a lot better. Increasing this further might result in even a better result. The training phase will need more time to be completed. However, it does not impact the speed of the testing phase. The extra time that the training phase cost can be neglected because all the models used need to be trained only once. The number of iterations should not be increased to the point that the model is overfitted. Increasing the iterations is listed as one of the future work options in section 7.2.

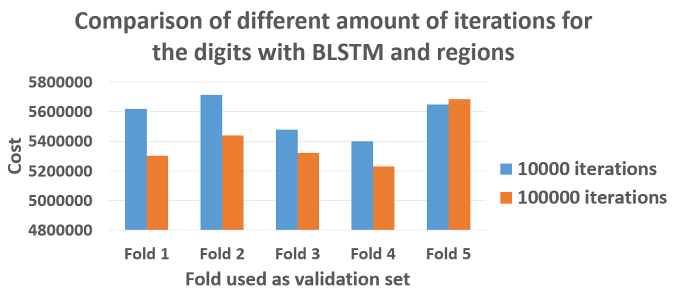


Fig. 13. Increasing the iterations to determine the dozens is in four out of the five folds a lot better.

5.3.8 Layers of cells. Sadoughi and Busso [38] use different cells and multiple cells of the same type. This paper researches speech animation, but the same concept can work for facial animation retargeting. Evaluating different types of cells was too time-consuming. Therefore, using a different amount of layers of basic lstm cells were explored. The results are visible in figure 14. The values used to create this figure can be found in table 15 in appendix B. This figure shows that using two layers of basic lstm cells is the best option. It also shows that using four cells is far worse than using three cells. Using more would probably not result in a better result than using two cells behind each other. Exploring this further with different cells as well listed as a future work option in section 7.2.

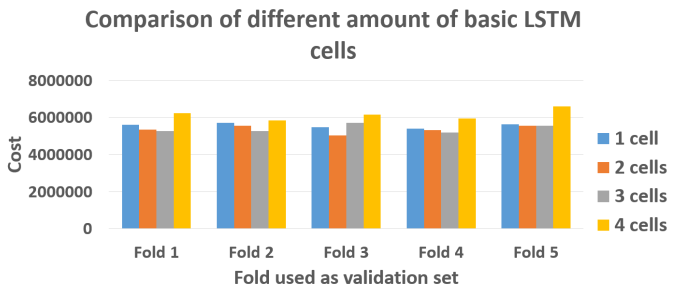


Fig. 14. The best option is using two basic LSTM cells. Increasing it above four most likely will not improve the results, since using four is much worse than three.

5.3.9 Best setup. Both experiments, increasing the iterations and increasing the number of cells, were evaluated separately. In theory, combining them should result in an even better result. Those results are in table 16 in appendix B. This test uses the same parameters as used in section 5.3.8.

Combining all the findings result in the following setup. Predicting the dozens and units both use two cell BLSTM with regions with a time step of 2, 700 hidden layers and a learning rate of 0.001. The prediction of the dozens also uses 100000 iterations and a batch size

		Angry	Frustrated	Happy	Neutral	Sad
RBFN	Eyes	77673,33863	51652,93689	69361,02799	29421,32481	70910,79507
RNN	Eyes	102449,1128	58550,80988	72916,66126	35702,55761	73517,10054
RBFN	Brows	170830,4391	76997,38767	69607,44765	33422,32102	48029,61052
RNN	Brows	171840,3323	78221,13787	80079,22977	35572,32923	38656,77095
RBFN	Mouth	452220,7416	223114,6833	389144,2244	315032,8154	99563,35595
RNN	Mouth	627923,4478	319158,947	506471,5337	486950,6468	190990,3878
RBFN	Total	1093599,338	597898,5932	766585,6658	565946,5197	409199,4966
RNN	Total	1340017,353	744407,1802	918582,3056	761780,5744	492076,3114

Table 3. A comparison of the testing results. These results are broken down in regions.

of 200. The unit prediction uses only 1000 iterations and a batch size of 100. Every combination of parameters has not been tested since this would take to much time. Therefore, parameters such as hidden layers were not reevaluated again. Besides this, most of the test done did not use hidden layer value of 700, just because it would cost to much time. Reassessing previously found parameters might be worthwhile. Therefore, it is listed as a future work option in section 7.2. To limit the time this idea costs some experiments can be skipped if it is obvious what happens. For example, reducing the batch size, which is done in section 5.3.6.

6 RESULTS

The best setting for RNN found is using the ceiling function to first predict the dozens. Predicting the dozens and units both use two cell BLSTM with regions with a time step of 2, 700 hidden layers and a learning rate of 0.001. The prediction of the dozens also uses 100000 iterations and a batch size of 200. The unit prediction uses only 1000 iterations and a batch size of 100. The experiments from this section are all run on Intel XEON E5-1620 3.50GHz computer with 16gb RAM and an NVIDIA Quadro K2200 4gb running Windows 7. The RNN experiments in this section are all run on the GPU since it is not realistic to run on the CPU for a movie or game due to the speedup running it on a GPU gives. Therefore, the RNN results used to create videos for the survey and to calculate cost this section are nondeterministic. Repeating these test will result in different results.

The cost function is used in section 5.3 to determine which parameter setup is best. This evaluation is also done for the results of RBFN and RNN. This is shown in section 6.1. Besides the cost results, a survey was held to determine how good the visual results are. The results are used to perform a student's t-test and determine if RBFN or RNN is significantly better or equal. This is done in section 6.2.

6.1 Cost results

The cost function used throughout this research first translates the blendshape weights back to motion capture data. The Euclidean distance between this and the actual motion capture data of the frame is calculated. This is done for each frame. The results of RBFN and RNN are in table 3. To compare RBFN and RNN the first and last predictions of RBFN are skipped because those are not predicted with RNN since a time step value of 2 was used to create these results. The results are split up in eyes, brows, mouth and total. These regions are the same as used in section 5.3.3. The markers used for each region can be found in table 1. The results of the eyes are in figure 15. These results show that for angry the eyes with RNN are a lot off compared to RBFN. The other four expressions are close but always in favour of RBFN. Looking closely at the visual results of RBFN, the eyelids move a tiny bit up and down rapidly. However, this is also in the motion capture data. RNN does not show this behaviour, which does lead to a higher cost calculated in this section. RNN has between 3.7%, for sad, and 31.6%, for angry, more cost for the eyes than RBFN.

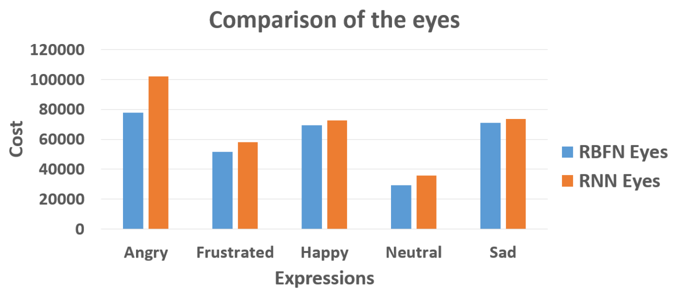


Fig. 15. A comparison of the eye region.

The results of the brows are in figure 16. Those results are close, RBFN is better for all the expressions except sad, there RNN is better. RNN has between 19.4% less cost, for sad, and 15.0%, for happy, more cost for the brows than RBFN.

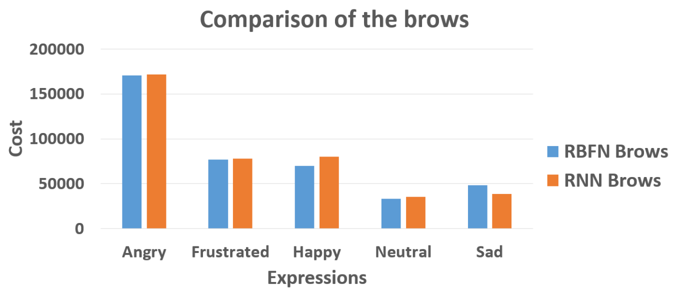


Fig. 16. A comparison of the brows region.

The results of the mouth are in figure 17. RNN is a lot off compared to RBFN in this area. A possible reason can be that for the blendshapes mouth smile, and mouth frown RBFN barely predicted

	Average	Standard deviation	n	T	p=5%
Angry	3,182795699	1,327510699	93	-5,936549425	RBFN significantly better
Frustrated	2,903225806	1,218711784	93	-8,67876017	RBFN significantly better
Happy	3,548387097	1,517144817	93	-2,870653526	RBFN significantly better
Neutral	3,836956522	1,40090944	92	-1,116316345	No significance
Sad	3,860215054	1,083516754	93	-1,244131388	No significance
Beast	3,477419355	1,50840862	155	-4,313205637	RBFN significantly better
Cartoon	3,561290323	1,315338156	155	-4,152461793	RBFN significantly better
Female	3,279220779	1,261295733	154	-7,091623851	RBFN significantly better
Total	3,439655172	1,371123332	464	-8,803144619	RBFN significantly better

Table 4. The results of a student's t-test for each expression and each character separately.

above 0, while RNN has the value almost always above 0. The mouth has the most significant difference of all the regions. This difference was also mentioned by one of the participants in the survey. RNN has between 29.8%, for happy, and 91.6%, for sad, more cost for the mouth than RBFN.

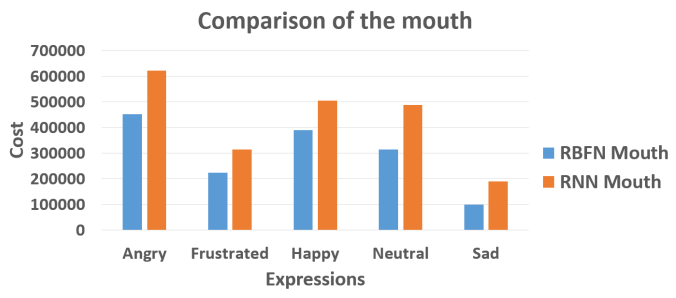


Fig. 17. A comparison of the mouth region.

In figure 18 the total cost is shown. The mouth can explain the majority of the difference between RBFN and RNN. All expressions show that RBFN is a lot lower in cost compared to RNN. RNN has between 19.6%, for happy, and 34.6%, for neutral, more cost for the whole face than RBFN.

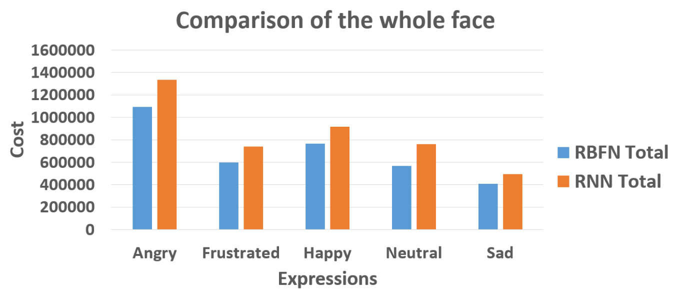


Fig. 18. A comparison of the whole face.

The time it takes to train all RNN models serial is so much more than RBFN. This is shown in table 17 in appendix B. The training time, however, can be optimised by training each blendshape parallel and each split model parallel as well. However, RNN is a lot quicker in the testing phase. With 100 seconds of testing data, RNN can close to real time. RNN needs a buffer because the current frame also needs the next frame to predict. RBFN is not in real time. However, again this process can be done parallel, and the normal frame rate of a movie is 24fps, while the testing set has 120 fps. Without optimising RBFN, it can do it real time as well, as long as the testing set is at 24 fps.

6.2 User experiment

A survey [50] was held to evaluate the visual results. A screenshot of the survey is shown in figure 23 in appendix D. This survey has 15 comparisons. Each comparison uses the visual results of RBFN and RNN of the same character with the same expression. The testing sets used have a sequence of 20 seconds of data and contain the extremes of the expression it represents. A seven-point Likert scale is used to determine which option is better and how much better it is. It was randomly decided if RNN was option 1 or option 2, RBFN was the other option. 31 people filled in the survey. One of the entries had a single question, female with the neutral test sequence, not filled in. The filled in part of that entry is still used in the evaluation part. Since it was randomly decided if RNN was option 1 or option 2, the results were adjusted so that for all the comparisons 7 means that RNN is a lot better, and 1 means that RBFN is a lot better. A one-sided student's t-test was done with $p = 0.05$. The resulting statistics can be found in table 4. The expressions angry, frustrated, and happy, showed that RBFN is significantly better. For neutral and sad, the difference is not significant. The averages are shown in figure 19.

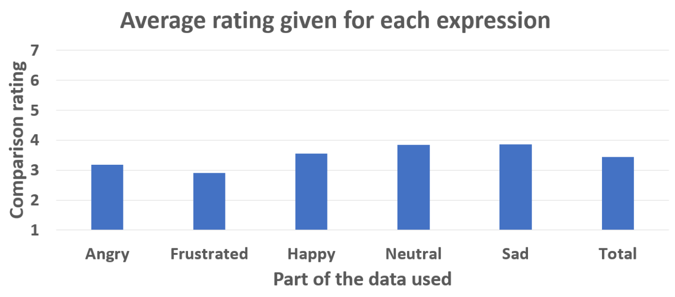


Fig. 19. A comparison of the ratings given by the participants for each expression. In contrast, the average of all the data is provided as well.

The results of each character show that RBFN is significantly better. The t-scores of beast and cartoon are about the same, while female differs a lot. So which character is used does influence the results. However, all character have the same outcome that RBFN is significantly better. The averages are shown in figure 20. Using all the data also indicates that RBFN is significantly better.

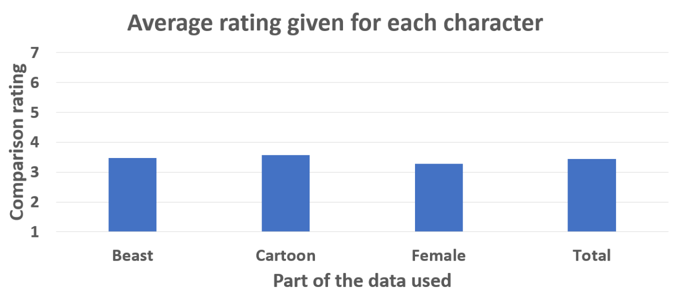


Fig. 20. A comparison of the ratings given by the participants for each character. In contrast, the average of all the data is provided as well.

7 DISCUSSION

RNN has its limitations. Even if it gets improved further, the limitations listed in section 7.1 will most likely still hold. The results in section 6 indicate that RBFN is significantly better. RNN has a lot to improve. However, in the present research not all improvements could be explored due to time limitations. Possible improvements are listed and discussed in section 7.2.

7.1 Limitations

The reason post-processing is necessary is that multiple frames have the same prediction and then there is a significant change to another sequence of the same predictions. The reason can be that with a time step of two, too many frames are too similar. Therefore the models used cannot differentiate between them. The training and testing input needs to be changed so the model can learn these subtle differences. Ideally, RNN should work without post-processing.

Another limitation is the time it takes. In the machine learning section, section 6.1, the time it takes to test it is discussed. The training time is not an issue, the time finding the optimal parameters is. CPU calculations are a lot slower and multiple, in this research five-fold cross-validation is used to evaluate each parameter adequately. Due to time constraints, not every combination was tested in this research. For example, the number of hidden units was evaluated at the beginning of the RNN section, section 5.3, but after changing to BLSTM and using more cells it was not reevaluated again. A different number of hidden units between 400 and 800 could be better than picking 700. RBFN only has one parameter, which is the kernel function. Finding the best kernel function is done a lot quicker than finding the best parameters for RNN. Continuing this research will require lots of time and CPUs.

7.2 Future Work

Testing different timesteps for LSTM, BLSTM and BLSTM with regions has been done in section 5.3.4. There the time step value 2 up to and including 6 were evaluated. An improvement with possible a huge impact is to increase the time step value above 6. BLSTM with a time step value of 11 is used in [48] for speech animation. If it works for speech animation, it is not a guarantee that it also works for facial animation retargeting. However, as shown in tables 10, 11 and 12 in appendix B and in figures 9, 10 and 11, time step 6 is better than time step 5. This observation could suggest that the current optima are just local and not global. Which is likely since a time step value of 2 only uses the previous frame and the next frame while using a bigger sequence should lead to bigger differences in the training data, which should help to learn better features.

A different setup of the cells could also lead to a better result. Using multiple basic LSTM cells has been evaluated in section 5.3.8. For example, [38] proposes two joint models with different types of cells. This research did only use basic lstm cells to experiment with to limit the number of parameters. However, there are a lot of different types of cells which can be used. Using different types of cells together could lead to significant improvements.

Changing the input to a lower dimension could also help the results. Using regions and blendshape-specific input has been evaluated in section 5.3.3. A possible approach which has not been tested is principal component analysis or PCA. PCA can change the input to a lower dimensional one while still preserving data which differentiates different frames. A previous experiment done for this research showed that PCA and RBFN did not outperform RBFN without PCA. However, it might be worth exploring for RNN. [20] first encodes the input and then decodes the output of RNN, which is a similar approach to using PCA. Encoding and decoding could also yield an improvement.

The RNN mouth region losses the most compared RBFN. This is shown in figure 17 in section 6.1. Having region specific or even blendshape specific parameters can lead to improvements. However, this is time-consuming since the number of parameters to optimise is multiplied by the number of regions or the number of blendshapes respectively.

In section 5.3.1, the predictions are split up to prevent certain blendshape weights not to be picked due to a low amount of representation. This problem could be solved by drawing the training set differently. Now each expression gets the same amount of training examples, while this does not represent necessarily cover all the possible blendshape weight values for each blendshape. Creating different training sets for each blendshape in which each blendshape weight value has the same amount of representation could fix the issue. Using all of the 101 possible blendshape weights in one prediction might be doable then. The problem of combining the optimal prediction with weak data, which can result in a less optimal prediction is then fixed. However, the amount of training data to pick from should be big enough for this to work. The training data without one of the five expressions could not be selected in the training sets used for this research, but with this idea, it can be used. However, predicting 101 possible blendshape weights in one model might be too much, but this idea gives it the best chance.

In this research, the cross-validation folds are drawn at random as described in section 5.3. Picking it random could, in theory, lead to an over or under-representation of an expression, which could lead to less optimal parameters. If each expression has 20% representation in each fold, then this issue could be solved. The impact of this change is not known, but it might results in a more stable result for each fold for different values. For example, in figure 7 in section 5.3.3, using regions is the best for three of the five folds and second for two out of the five folds. Using folds which are equally divided over the expressions might result in all five folds showing that regions is better than the alternatives.

Lowering the batch size, which was explored in section 5.3.6 did not yield a better result. The reason might be because reducing the batch size without increasing the iterations might lead to overfitting. It would be interesting to see what the results are when increasing the iterations enough, so that batch size times number of iterations is the same for each setup.

Some small improvement might be changing the learning rate, increasing the iterations for both the digit, which is done in section 5.3.7, and unit prediction and increasing the batch size.

8 CONCLUSION

The visual results in section 6.2 showed that RBFN performs significantly better, except for neutral or sad expressions. Machine learning results in section 6.1 show that RBFN has a lower cost in all regions, with the most significant difference being in the mouth region. So RBFN is better than the best RNN setup in this research. However, there are no papers published using RNN for facial animation retargeting. And thus this research is just finding out if it has potential. While the best RNN setup discussed in this research is not sufficient for daily practice with improvements, it might be. The suggestions in section 7.2 work in other fields, which does not necessarily mean that they do work for facial animation retargeting, but there is enough overlap to assume that it has potential. The research question, which is: "Does a recurrent neural network to learn the mapping from an actor's face to a virtual model result in a better result than radial basis functions?", could not be disproven by these results because of the amount of possible parameter options to try out. I do believe that with future research RNN has enormous potential. The training time will most likely increase. However, each model only needs to be trained once, which means that the training time can be neglected. If the results improve enough, RNN could be used in movies and games.

ACKNOWLEDGMENTS

I want to thank my supervisor, dr. Zerrin Yumak, for her guidance and feedback during this thesis project. Also, I would like to thank my second supervisor, prof. dr. Remco Veltkamp, for his time, to evaluate this thesis project.

I want to thank Cas van Cooten for proofreading this thesis. Furthermore, I want to thank tofusan for the "anime head" slider used to create the cartoon character. Last but not least, I would like to thank the participants in my survey for there time and the valuable data they provided me.

REFERENCES

[1] Ken Anjyo, Hideki Todo, and JP Lewis. 2012. A practical approach to direct manipulation blendshapes. *Journal of Graphics Tools* 16, 3 (2012), 160–176.

[2] Vincent Barrielle, Nicolas Stoiber, and Cédric Cagniart. 2016. Blendforces: A dynamic framework for facial animation. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 341–352.

[3] Philippe Bergeron and Pierre Lachapelle. 1985. Controlling facial expressions and body movements in the computer generated animated short\'Tony de Peltrie\'. (1985).

[4] Bernd Bickel, Mario Botsch, Roland Angst, Wojciech Matusik, Miguel Otaduy, Hanspeter Pfister, and Markus Gross. 2007. Multi-scale capture of facial geometry and motion. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 33.

[5] Christopher M Bishop. 2006. Machine learning and pattern recognition. *Information Science and Statistics. Springer, Heidelberg* (2006).

[6] Sofien Bouaziz and Mark Pauly. 2014. *Semi-Supervised Facial Animation Retargeting.* Technical Report.

[7] Sofien Bouaziz, Yangang Wang, and Mark Pauly. 2013. Online modeling for realtime facial animation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 40.

[8] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. 2002. Turning to the masters: motion capturing cartoons. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 399–407.

[9] David S Broomhead and David Lowe. 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks.* Technical Report. Royal Signals and Radar Establishment Malvern (United Kingdom).

[10] Ian Buck, Adam Finkelstein, Charles Jacobs, Allison Klein, David H Salesin, Joshua Seims, Richard Szeliski, and Kentaro Toyama. 2000. Performance-driven hand-drawn animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering.* ACM, 101–108.

[11] Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N Chang, Sungbok Lee, and Shrikanth S Narayanan. 2008. IEMOCAP: Interactive emotional dyadic motion capture database. *Language resources and evaluation* 42, 4 (2008), 335.

[12] Erika Chuang and Chris Bregler. 2002. Performance driven facial animation using blendshape interpolation. *Computer Science Technical Report, Stanford University* 2, 2 (2002), 3.

[13] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning.* ACM, 160–167.

[14] Timothy Costigan, Mukta Prasad, and Rachel McDonnell. 2014. Facial retargeting using neural networks. In *Proceedings of the Seventh International Conference on Motion in Games.* ACM, 31–38.

[15] Zhen Cui, Shiguang Shan, Haihong Zhang, Shihong Lao, and Xilin Chen. 2012. Image sets alignment for video-based face recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2626–2633.

[16] Aymeric Damien. 2017. Recurrent neural network for the MNIST dataset. https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/recurrent_network.py

[17] Zhigang Deng, Pei-Ying Chiang, Pamela Fox, and Ulrich Neumann. 2006. Animating blendshape faces by cross-mapping motion capture data. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games.* ACM, 43–48.

[18] Ludovic Dutreve, Alexandre Meyer, and Saïda Bouakaz. 2008. Feature points based facial animation retargeting. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology.* ACM, 197–200.

[19] Ke Fan, Ajmal Mian, Wanquan Liu, and Ling Li. 2016. Unsupervised manifold alignment using soft-assign technique. *Machine Vision and Applications* 27, 6 (2016), 929–942.

[20] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision.* 4346–4354.

[21] Fahimeh Ghasemi, Afshin Fassihi, Horacio Pérez-Sánchez, and Alireza Mehri Dehnavi. 2017. The role of different sampling methods in improving biological activity prediction using deep belief network. *Journal of computational chemistry* 38, 4 (2017), 195–203.

[22] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. 2009. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence* 31, 5 (2009), 855–868.

[23] Mark Henne, Hal Hickel, Ewan Johnson, and Sonoko Konishi. 1996. The making of toy story [computer animation]. In *Compcon'96.'Technologies for the Information Superhighway'Digest of Papers.* IEEE, 463–468.

[24] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[25] Alexandru Eugen Ichim, Sofien Bouaziz, and Mark Pauly. 2015. Dynamic 3D avatar creation from hand-held video input. *ACM Transactions on Graphics (ToG)* 34, 4 (2015), 45.

[26] Alexandru Eugen Ichim, Ladislav Kavan, Merlin Nimier-David, and Mark Pauly. 2016. Building and animating user-specific volumetric face rigs.. In *Symposium on Computer Animation.* 107–117.

[27] Pushkar Joshi, Wen C Tien, Mathieu Desbrun, and Frédéric Pighin. 2006. Learning controls for blend shape based realistic facial animation. In *ACM Siggraph 2006 Courses.* ACM, 17.

[28] Natasha Kholgade, Iain Matthews, and Yaser Sheikh. 2011. Content retargeting using parameter-parallel facial layers. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 195–204.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[30] Manfred Lau, Jinxiang Chai, Ying-Qing Xu, and Heung-Yeung Shum. 2009. Face poser: Interactive modeling of 3d facial expressions using facial priors. *ACM Transactions on Graphics (TOG)* 29, 1 (2009), 3.

[31] Hao Li, Thibaut Weise, and Mark Pauly. 2010. Example-based facial rigging. In *Acm transactions on graphics (tog)*, Vol. 29. ACM, 32.

[32] Faezeh Movahedi, James L Coyle, and Ervin Sejdić. 2017. Deep belief networks for electroencephalography: A review of recent contributions and future outlooks. *IEEE journal of biomedical and health informatics* (2017).

[33] Jun-yong Noh and Ulrich Neumann. 2001. Expression cloning. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 277–288.

[34] Verónica Costa Orvalho, Ernesto Zacur, and Antonio Susin. 2008. Transferring the rig and animations from a character to different face models. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1997–2012.

[35] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[36] Frederick I Parke. 1972. Computer generated animation of faces. In *Proceedings of the ACM annual conference-Volume 1*. ACM, 451–457.

[37] Yuru Pei, Fengchun Huang, Fuhao Shi, and Hongbin Zha. 2012. Unsupervised image matching based on manifold alignment. *IEEE transactions on pattern analysis and machine intelligence* 34, 8 (2012), 1658–1664.

[38] Najmeh Sadoughi and Carlos Busso. 2017. Joint learning of speech-driven facial motion with bidirectional long-short term memory. In *International Conference on Intelligent Virtual Agents*. Springer, 389–402.

[39] Jun Saito. 2013. Smooth contact-aware facial blendshapes transfer. In *Proceedings of the Symposium on Digital Production*. ACM, 7–12.

[40] Haşim Sak, Andrew Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[41] Jaewoo Seo, Geoffrey Irving, JP Lewis, and Junyong Noh. 2011. Compression and direct manipulation of complex blendshape models. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 164.

[42] Yeongho Seol, JP Lewis, Jaewoo Seo, Byungkuk Choi, Ken Anjyo, and Junyong Noh. 2012. Spacetime expression cloning for blendshapes. *ACM Transactions on Graphics (TOG)* 31, 2 (2012), 14.

[43] Yeongho Seol, Wan-Chun Ma, and JP Lewis. 2016. Creating an actor-specific facial rig from performance capture. In *Proceedings of the 2016 Symposium on Digital Production*. ACM, 13–17.

[44] Yeongho Seol, Jaewoo Seo, Paul Hyunjin Kim, JP Lewis, and Junyong Noh. 2011. Artist friendly facial animation retargeting. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 162.

[45] Greg Singer. 2003. The two towers: Face to face with gollum. *Animation World Network* 1, 2 (2003).

[46] Jaewon Song, Byungkuk Choi, Yeongho Seol, and Junyong Noh. 2011. Characteristic facial retargeting. *Computer Animation and Virtual Worlds* 22, 2-3 (2011), 187–194.

[47] Robert W Sumner and Jovan Popović. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 399–405.

[48] Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. 2017. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 93.

[49] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*. 2643–2651.

[50] Teus van Oosterom. 2018. Survey for Facial Animation Retargeting. https://formview.io/#/wzthkjcyxhlimdw/survey?header=1

[51] CJ Van Rijsbergen. 1979. Information retrieval. dept. of computer science, university of glasgow. *URL: citeseer. ist. psu. edu/vanrijsbergen79information. html* 14 (1979).

[52] Chang Wang and Sridhar Mahadevan. 2009. Manifold Alignment without Correspondence.. In *IJCAI*, Vol. 2. 3.

[53] Chang Wang and Sridhar Mahadevan. 2011. Heterogeneous domain adaptation using manifold alignment. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22. 1541.

[54] Yang Wang, Xiaolei Huang, Chan-Su Lee, Song Zhang, Zhiguo Li, Dimitris Samaras, Dimitris Metaxas, Ahmed Elgammal, and Peisen Huang. 2004. High Resolution Acquisition, Learning and Transfer of Dynamic 3-D Facial Expressions. In *Computer Graphics Forum*, Vol. 23. Wiley Online Library, 677–686.

[55] Chung-Hsien Wu, Jen-Chun Lin, and Wen-Li Wei. 2014. Survey on audiovisual emotion recognition: databases, features, and data fusion strategies. *APSIPA transactions on signal and information processing* 3 (2014).

[56] Feng Xu, Jinxiang Chai, Yilong Liu, and Xin Tong. 2014. Controllable high-fidelity facial performance transfer. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 42.

[57] Eduard Zell, JP Lewis, Junyong Noh, Mario Botsch, et al. 2017. Facial retargeting with automatic range of motion alignment. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 154.

A PIPELINE

The pipeline used is shown in figure 21. First, the dataset is preprocessed to remove outliers. Details of this process are written in section 4.1. Then 20 seconds testing sequences were found for each expression. The training sets may not include the testing data. The dataset, therefore, removes the testing sets from its data. Both training sets pick their data from the remaining dataset. For RBFN 20 random frames for each expression are selected, which results in a training set of 100 frames. For RNN 20000 sequences of 11 frames are randomly chosen for each expression, which leads in a training set of 100000 frames. The 11 frames of the sequence all have the same expression given by the human evaluation done for the IEMOCAP dataset. Both the RBFN and RNN model are trained with there training sets and then evaluated by the testing sequences. The results of RNN are then post-processed. Details of this process are written in section 4.2. The visual results are made with a unity script reading the blendshape weights in and returning the resulting images. For the machine learning results, predicted blendshape weights are translated back to motion capture values. The Euclidian distance between those values and the actual values are used to determine the cost.

A.1 RNN

The base of the code for the training and testing of an RNN is the tensorflow example to predict numbers out of the MNIST dataset. [16] is that example. Multiple changes were made. In the function RNN, different models can be built with cells. For each blendshape first regional data to use for the blendshape is defined. The blendshape weights corresponding to the dataset are multiplied by 100. Then the training set for the first prediction is made. This set contains the motion capture data with the correct dozens prediction. A forward prediction and backwards prediction is done which results in two lists of arrays with odds for each sequence used as testing set. The two lists are used to create one, by multiplying the forwards predicted array with the corresponding backwards predicted array. For each array in the resulting list, the prediction is the blendshape weight with the highest value. By using this prediction, the testing set is split up to predict units separated for each different prediction. The dozen predictions times ten is written down as the blendshape weights. Predicting the units is similar to predicting the dozens, only a smaller training and testing set is used. Therefore, the amount of iterations used is also lower. If the dozen prediction is 0, the unit prediction does not have to happen because the only possible unit prediction is 0. The unit predictions minus nine is deducted from the blendshape weights. In the end, the blendshape weights are divided by 100 and written away. The pseudocode is shown in algorithm 1.

The tensorflow library is used. This library is used for the predictions. Numpy is used to transform data to matrix or array shape. The math library is used for the ceil function. The shuffle function from sklearn is used. Itertools is used to combine the four folds used as training set. To for tensorflow to use the CPU os is used. The time calculation done in section 6.1 is done with the time library.

ALGORITHM 1: Algorithm used to create testing results.

```
Data: Training set, which contains motion capture data and blendshape weights of the "current" frame for each sequence. Testing set, which contains
      motion capture data.
Result: The blendshape weights of the testing sets.
Read in the data
foreach Blendshape do
    if Regional blendshape then
        Reduce dimensions of the input
    end
    Ceil function to create first and second prediction training sets
    Train model with current and future frames of the first prediction training set
    Make forwards predictions for the testing set
    Train model with current and previous frames of the first prediction training set
    Make backwards predictions for the testing set
    Multiply forwards and backwards predictions
    Define prediction for each frame as argmax
    Split the testing set up for the second prediction
    foreach second prediction do
        Train model with current and future frames of this second prediction training set
        Make forwards predictions for this second prediction testing set
        Train model with current and backwards frames of this second prediction training set
        Make backwards predictions for this second prediction testing set
        Multiply forwards and backwards predictions
        Define prediction for each frame as argmax
        Define final prediction as first prediction - (second prediction - 9)
    end
end
Write blendshape weights away
```

A.2 Unity script

The Unity script expects blendshape weights between 0 and 1. The blendshapes in Unity have a range of 0 up to and including 100. So first the script read the blendshape weights and multiplies them by 100. Then for each frame, the blendshape weights are set for each blendshape, and an image of the face is written away.

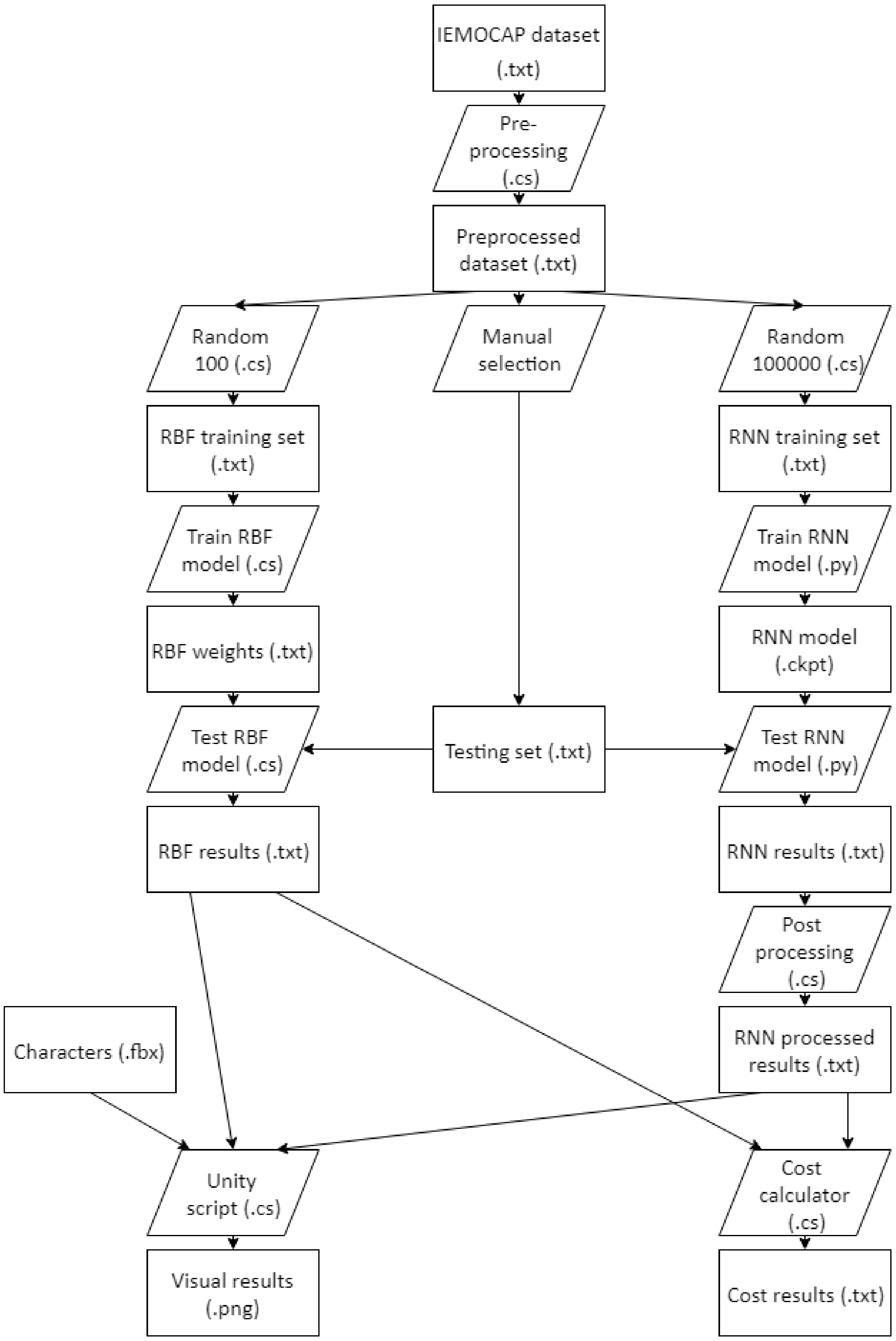


Fig. 21. The pipeline used to get the visual and machine learning results of RBFN and RNN.

B TABLES

Blendshape weight	Basic	Ceil	Floor	Round
0	1048651	891802	730522	753237
1	5703	18280	75674	41483
2	6841	47399	29571	40208
3	6167	46042	3018	34635
4	10868	0	0	0
5	8768	55480	46268	35125
6	7062	0	0	3284
7	8390	55588	67027	4607
8	10663	0	0	0
9	6974	13159	8020	26881

Table 5. The number of predictions for blendshape weight 0 up to and including 9 for each RNN approach.

	Min/max each value	Min/max each xyz	Random 100	Random 200	Random 300
Angry	1075853,125	1158106,016	1093600,197	1078826,853	1076569,469
Frustrated	614017,7993	638925,8708	597899,0464	592974,666	592989,3142
Happy	724800,8891	781465,8081	766586,2642	746980,7439	743279,3276
Neutral	566441,1547	627773,0563	565947,0097	560307,1253	562446,5585
Sad	433012,3712	508619,5069	409199,9949	404973,4897	406622,3582
Total	3414125,339	3714890,258	3433232,512	3384062,878	3381907,027

Table 6. A comparison of different training set selection methods.

	Basic	Ceiling	Floor	Round
Fold 1	7780604,13	5683466,318	5436843,346	5792150,725
Fold 2	7900003,796	5704089,709	6541804,502	5850025,553
Fold 3	7655038,127	5492787,199	6302987,448	5812551,238
Fold 4	6593696,139	5705279,616	6125270,35	5786614,26
Fold 5	7239392,751	5864771,671	5789465,367	5589391,962
Total	37168734,94	28450394,51	30196371,01	28830733,74

Table 7. A comparison of predicting 101 possible blendshape weights in one model and dividing it up into multiple predictions.

	400 hidden layers	500 hidden layers	600 hidden layers	700 hidden layers	800 hidden layers
Fold 1	5683466,318	5562748,952	5436860,988	5482060,887	5556786,243
Fold 2	5704089,709	5615390,359	5640736,919	5688504,649	5496774,978
Fold 3	5492787,199	5349832,518	5358872,683	5290387,419	5393112,394
Fold 4	5705279,616	5507647,53	5571736,256	5419783,259	5442388,032
Fold 5	5864771,671	5620407,067	5851317,915	5713453,495	5925085,587
Total	28450394,51	27656026,43	27859524,76	27594189,71	27814147,23

Table 8. Tuning the hidden layer parameter.

	Base	Regions	Blendshape specific
Fold 1	5562748,952	5403190,484	5410159,35
Fold 2	5615390,359	5517102,978	5610214,667
Fold 3	5349832,518	5375696,701	5397162,133
Fold 4	5507647,53	5417700,64	5344646,737
Fold 5	5620407,067	5463940,094	5484514,229
Total	27656026,43	27177630,9	27246697,12

Table 9. A comparison of using all the data, region specific data or blendshape specific data.

	Time step=2	Time step=3	Time step=4	Time step=5	Time step=6
Fold 1	6016944,853	5895530,354	6007078,608	6602804,194	6263493,461
Fold 2	6232611,156	6321872,266	6503549,945	5827015,801	6241775,428
Fold 3	5948840,051	6049797,286	5974335,737	6596679,714	6130323,064
Fold 4	5694297,531	5894625,699	5914172,436	5962720,481	5815465,337
Fold 5	6776259,75	6451483,547	6271859,689	6710450,312	6507899,614
Total	30668953,34	30613309,15	30670996,41	31699670,5	30958956,9

Table 10. A comparison of different time steps with LSTM.

	Time step=2	Time step=3	Time step=4	Time step=5	Time step=6
Fold 1	5634272,424	5678768,353	5721760,22	5802365,025	5977380,529
Fold 2	6201277,688	6226896,96	6519262,284	6144524,037	5950013,954
Fold 3	5873704,137	5776093,303	5947293,242	6169526,911	5729378,948
Fold 4	5968738,607	5702272,64	5935655,741	6051209,404	6219247,987
Fold 5	6414541,918	5871320,809	5850470,847	6420663,322	6049225,055
Total	30092534,77	29255352,06	29974442,33	30588288,7	29925246,47

Table 11. A comparison of different time steps with BLSTM.

	Time step=2	Time step=3	Time step=4	Time step=5	Time step=6
Fold 1	5619324,528	6029933,583	5402269,82	5469583,799	5649189,306
Fold 2	5712948,795	5861367,583	5863302,891	5829533,647	5690808,113
Fold 3	5480201,69	5483060,62	5783544,931	5781741,041	5486707,529
Fold 4	5401026,976	5460180,882	5994428,755	5761764,554	5651617,831
Fold 5	5648212,773	5479994,216	5636433,633	5922353,468	5782499,657
Total	27861714,76	28314536,88	28679980,03	28764976,51	28260822,44

Table 12. A comparison of different time step with BLSTM and regions combined.

	Timestep=2	Timestep=3	Timestep=4	Timestep=5	Timestep=6
20	33926263	34659612	34219423	34712443	33799697
100	28823660	29113616	28388691	29917383	29402264
200	27861715	28314537	28679980	28764977	28260822

Table 13. A comparison of different batch sizes with BLSTM and regions.

	10000 iterations	100000 iterations
Fold 1	5619324,528	5302920,264
Fold 2	5712948,795	5438516,426
Fold 3	5480201,69	5322965,262
Fold 4	5401026,976	5229617,439
Fold 5	5648212,773	5684327,798
Total	27861714,76	26978347,19

Table 14. A comparison of using more iterations to determine the dozens with BLSTM and regions.

	1 cell	2 cells	3 cells	4 cells
Fold 1	5619324,528	5348523,833	5275728,669	6251702,739
Fold 2	5712948,795	5555850,546	5277182,649	5848273,384
Fold 3	5480201,69	5048976,243	5718830,63	6154561,412
Fold 4	5401026,976	5328805,524	5195623,332	5953054,394
Fold 5	5648212,773	5570075,342	5562368,859	6603392,139
Total	27861714,76	26852231,49	27029734,14	30810984,07

Table 15. A comparison of different amount of basic LSTM cells.

	Best
Fold 1	5308769,844
Fold 2	5422952,1
Fold 3	5424605,167
Fold 4	5310002,316
Fold 5	5532376,431
Total	26998705,86

Table 16. This is the best option found in this research.

	Training time	Testing time
RBFN	00:00:03.717	00:06:31.997
RNN	19:37:59.991	00:00:36.605

Table 17. A comparison of the amount of time needed to train and test each approach.

C MARKER PLACEMENTS

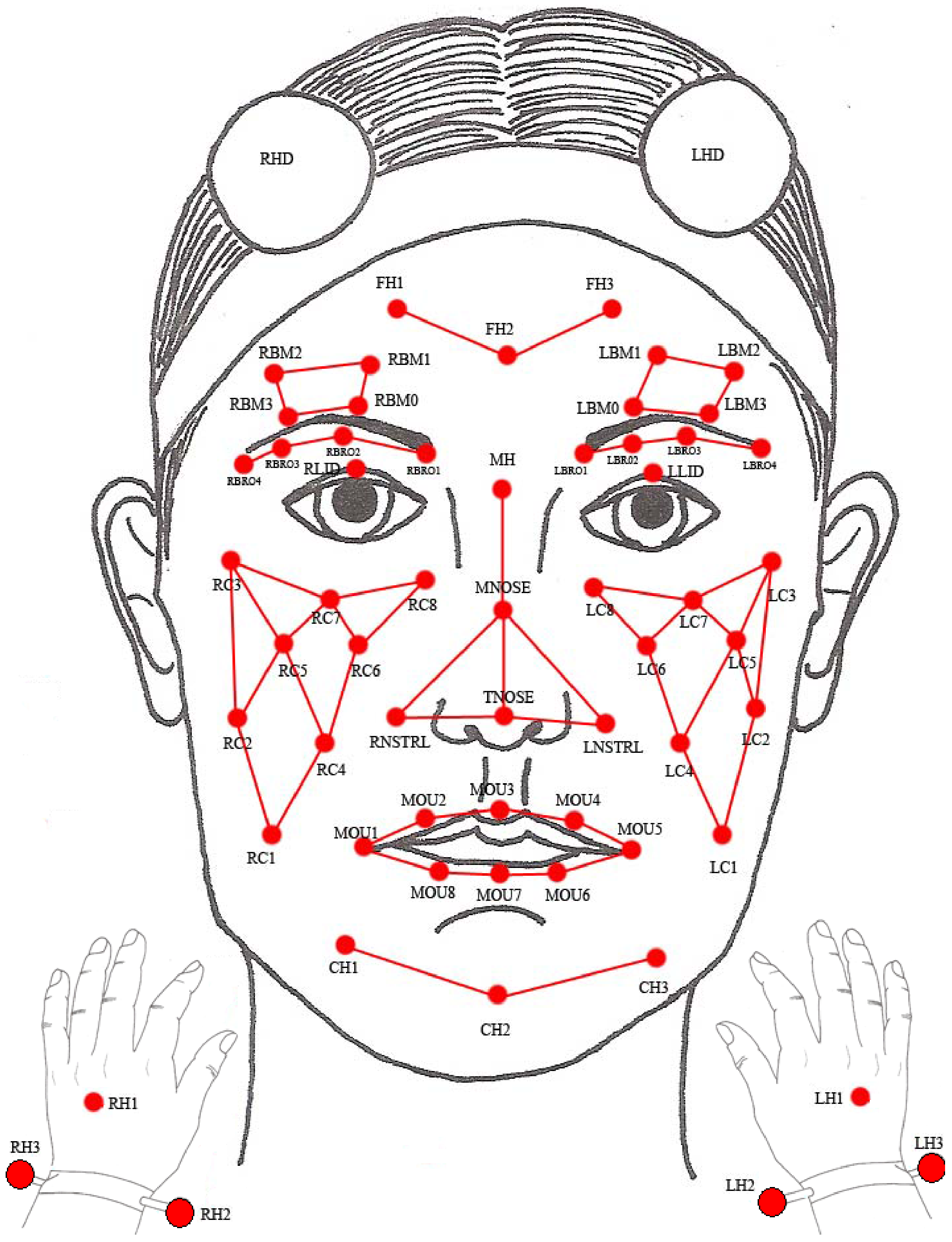
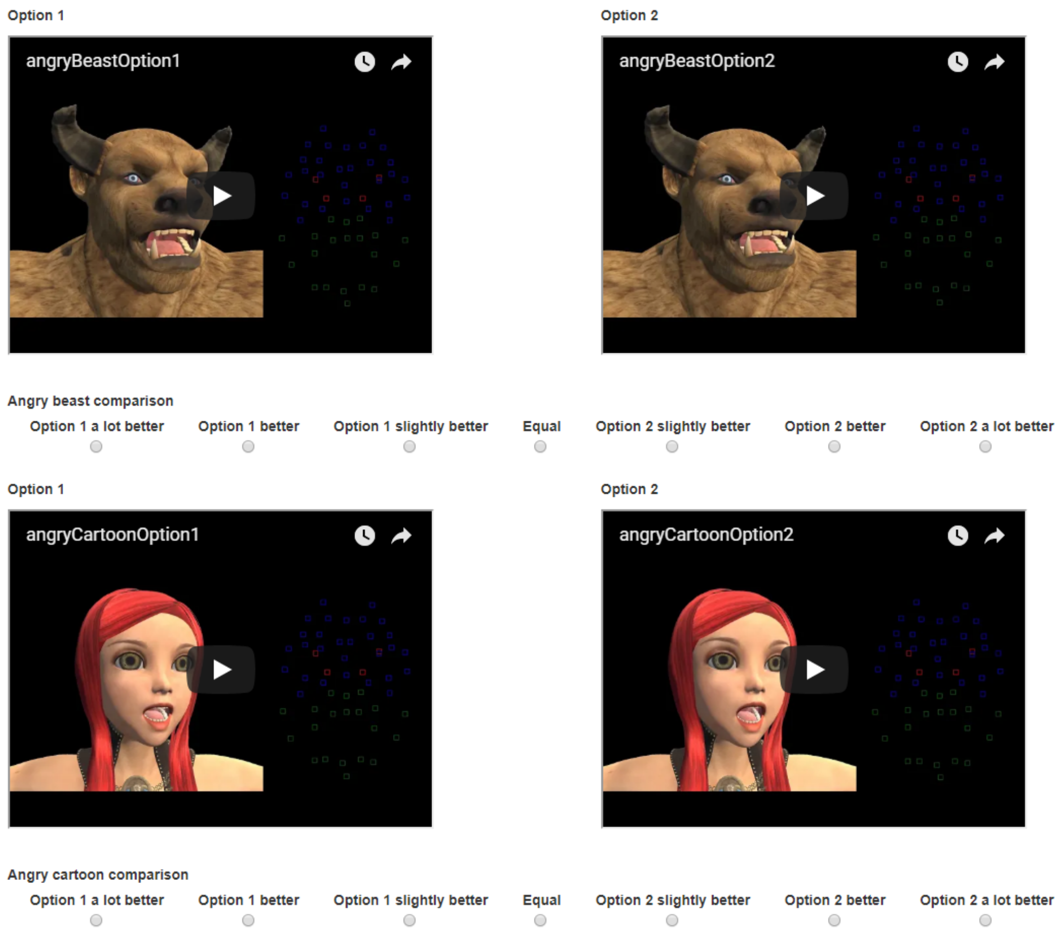


Fig. 22. The facial markers used for the IEMOCAP dataset.

D SURVEY SCREENSHOT



Option 1

angryBeastOption1

Option 2

angryBeastOption2

Angry beast comparison

Option 1 a lot better	Option 1 better	Option 1 slightly better	Equal	Option 2 slightly better	Option 2 better	Option 2 a lot better
○	○	○	○	○	○	○

Option 1

angryCartoonOption1

Option 2

angryCartoonOption2

Angry cartoon comparison

Option 1 a lot better	Option 1 better	Option 1 slightly better	Equal	Option 2 slightly better	Option 2 better	Option 2 a lot better
○	○	○	○	○	○	○

Fig. 23. A screenshot of the first two comparisons in the survey used for section 6.2.