Prediction of Gender from Responses to Real and Fake Smiles
An Application of Bidirectional Neural Nets and Genetic Algorithms

Zal Bhathena[1]

Australian National University, Australia, Canberra, ACT 0200 u6887931@anu.edu.au

Abstract. This work is a preliminary study on investigating the relationship between human responses to real and fake smiles. We analyze the pupillary responses of 10 Asian observers (6 male and 4 female) to real and fake smiles. Our analysis attempts to determine the gender of the observer from the pupillary responses to both real and fake smiles by training two types of neural networks: the first being a feed forward network and the second a bidirectional neural network. We use both random search and genetic algorithms for hyperparameter tuning, finding improved results with the latter approach. Several promising models with positive results were trained but overall, our results are inconclusive due to the limited size of the data set but encourage future study on a larger dataset and further experimentation.

Keywords: Neural Network · Genetic Algorithms · Pupillary Response · Random Search Bidirectional Neural Network · Fake vs Real Smiles · time series

1 Introduction

Human smiles are a complex social feature that elicit different responses from individuals in different contexts. Smiles can be used to display happiness or as a social signal. Hossain, Gedeon, Sankaranarayana, Apthorp, and Dawel not only show that genuine smiles (meaning a true display of joy, the underlying emotion) evoke different pupillary responses than fake ones but also that the pupillary response is opposite for males and females [1].

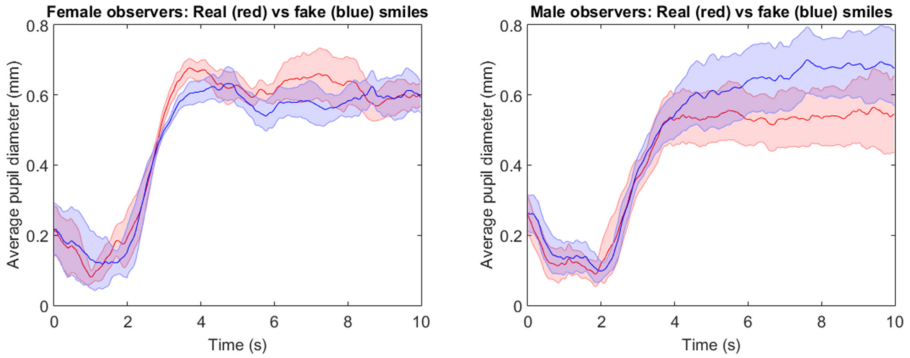

Fig. 1: Average pupil diameter timelines for all stimuli over female and male observers [1].

This leads to the question of whether this pupillary response generalizes across the population of male and female humans and whether it is possible to predict the gender of a person solely based on the response to a fake and real smile. Our goals are to determine if it is possible to identify the gender of an individual observer after showing them images of fake and real smiles. Using the data from the experiments run by Hossain, Gedeon, Sankaranarayana, Apthorp, and Dawel, we will train several models whose methodology can be used for future work. Given the small sample of data, we do not expect any statistically significant or meaningful results from the models themselves.

2 Dataset

The dataset we will be examining was constructed in a study by Hossain, Gedeon, Sankaranarayana, Apthorp, and Dawel. Pupillary data from 6 asian males and 4 asian women was recorded from observing videos of people exhibiting fake and real smiles. The recorded data is in time series format with 540 data points over 10 seconds for each observation.

In addition, some summary statistics for each observation are provided. From these, we can gain some insights into the differences of the pupillary responses of males and females. The average pupillary dilation for males for real smiles was .419 mm and for fake smiles was .490 mm, while for females it was .500 mm for real smailes and .473 mm for fake smiles. This is in accordance with the authors' findings that the male response for fake smiles was a higher pupillary dilation while for females it was a lower level of pupillary dilation. However, we see that for males the standard deviation of pupillary responses was .205 mm and for females was .204 mm. This means that although on average there is a difference in pupillary responses, differences for individuals will be obscured by the high levels of variance. A simple model based on comparing averages between the responses to fake and real smiles may not be enough to determine the gender of a subject.

3 Methods

Given the complex nature of pupillary data, the approach we will take is to train two types of neural networks that can classify individuals as either male or female based on pupillary data of the observer. The first will be a simple feed forward network while the second will be a bidirectional network.

3.1 Preprocessing

Before training any models, we must first prepare our inputs for ingestion. Our first step is to normalize our time series inputs, as it has been shown that this can significantly improve classification performance [2]. For our example, each input into the model will be a pair of pupillary responses for fake and real smiles for a single individual. Since different individuals may have different magnitudes of responses, we will normalize the range of responses of an individual to real values between 0 and 1. Note that the fake and real observations for an individual will be normalized independently.

Second, we will reduce the number of dimensions of our time series input data. It has been shown that computational complexity and often the generalizability of a network will increase if the dimensionality of input data is reduced [3]. In our case, the dimensions (the number of data points in for the time series describing pupillary dilation for each observation) of the pupillary data can be reduced by pooling over time. By summing dilation values over each 10 time step window, the dimensionality is reduced 10 fold. This is possible due to the low variance of data points that have temporal locality.

Finally, we need to preprocess the statistical feature data that has been provided for this dataset. Since there are only two data points for each individual for each statistic (one for the real and one for the fake smile time series), it does not make sense to normalize. Also, since we are doing cross validation, normalizing over all the individuals could potentially leak information about the missing items to the network, so we choose to leave these inputs as is.

3.2 Neural Network Architecture

Let us now discuss the architecture of the first networks. We will use two basic feed forward neural networks: one which is more traditional with one input layer and one hidden layer, while the other is only a slight modification to the traditional feed forward network. Each of these will be using sigmoidal activation functions (a reasonable choice for classification) and trained using binary cross entropy loss. From Fig. 2, we can see that the second network is simply two feed forward networks that feed into a final classifier network. The upper network takes the real smile pupillary data as an input, the lower network takes the fake smile pupillary data as an input and the final layer classifies the individual as either male or female. Finally, we will also experiment with different inputs. We will train using only time series pupillary data, only statistical data, and a combination of both.

3.3 Sample Size and LOOCV

One difficulty in training our network is our obvious lack of sample data. With data for only 10 individuals, determining the networks' true accuracy will be quite difficult and without proper safeguards, our network will also be susceptible to overfitting. Thus we need to leverage techniques to not only best maximize the data we have, but also allow our network to generalize.

Given a large dataset, one approach is to create 3 partitions of the data: a training set to train the model, a validation set to tune hyperparameters and a test set to determine the model performance on unseen data. In our case if we take this approach, our validation set and training set will be much too small, so instead we will use leave-one-out cross validation (LOOCV). This approach divides the dataset into only two partitions. The first is

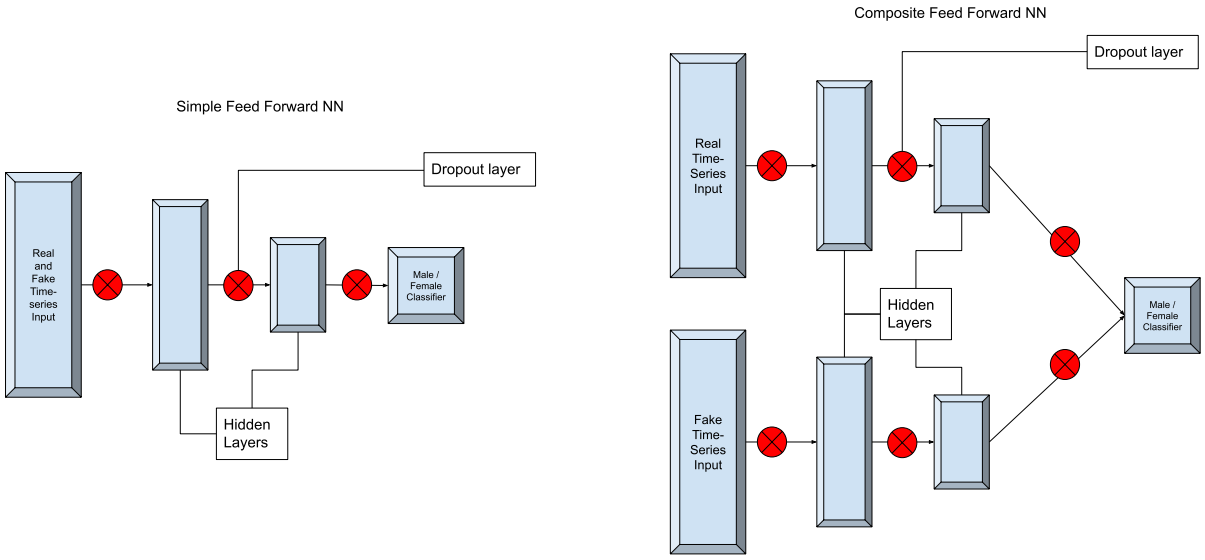

Fig. 2: The two feed forward neural network architectures used for classification

used for the LOOCV algorithm, while the latter is our standard test set. A single input example is now selected from the first partition to be our validation set while the model is trained on the rest of the data. This process of selecting elements repeated for every element in the model and the model evaluations are averaged over all of the validation runs. It has been shown that this approach gives at least as good of performance as the typical training/validation/test set split and in practice will out perform [4]. The trade off usually is the computational complexity of the approach but given the minute size of our data set, this is not a factor. In our implementation, the dataset is split into a partition of 8 individuals for LOOCV, and a test set of one male and one female.

The second issue of having such a small data set is overfitting. One well known technique for regularization of a network (the penalization of complexity) is dropout. Dropout is a technique that has shown to be state of the art and can help prevent neural networks from heavily depending on the activation of only several neurons. By clearing the weights of some proportion of neurons with some likelihood during training, the model will not depend too heavily on particular neurons and is shown to reduce overfitting [5].

3.4 Hyperparameter search

With the structure of our model described, all that remains is the tuning of the hyperparameters. There are several approaches that can be taken such as grid search or manual search, but we have chosen random search. This technique takes a stochastic approach and randomly selects a set of samples for the hyperparameters and tests each of them on the validation set. It has been shown that this approach, compared to grid and manual search, finds better or equivalent hyperparameters with much better computational complexity [6].

The hyperparameters that will need tuning for our network are the following: dropout rate, learning rate for optimizer (we will be using Adam) and the sizes of both hidden layers. The ranges for each of these are described in Table 1. Note the sizes of our network layers are rather constrained. Since we are dealing with a dataset with a very small number of samples, we decided to keep the range of network sizes rather constrained in order to reduce overfitting.

Table 1: The hyperparameters and ranges used for random search and the genetic algorithm.

Neural Network Hyperparameters	Lower Bound	Upper Bound	Model
Learning Rate	.00001	.01	All
Dropout Rate	.25	.75	All
First Hidden Layer Size	Fourth of input size	Half of input size	All
Second Hidden Layer Size	2	Half of previous hidden layer	Composite Feedforward
Backwards Layer Max Epochs	200	1000	Bidirectional

3.5 Genetic Algorithms

In addition to random search, we also explore the application of genetic algorithms for both feature selection and hyperparameter tuning. Genetic algorithms have been an industry standard technique allowing for major speedups in finding optimal hyperparameters over strategies such as grid search, random forest and other techniques [7]. Due to time constraints and limited computational resources, it is unlikely that we will be able to find optimal hyperparameters from random search. Thus we will train two sets of models using random search and genetic algorithms for hyperparameter optimization in hopes of finding a better set of hyperparameters with genetic algorithms.

For our hyperparameter optimization, we choose the same set of hyperparameters from Table 1 as our genetic features for our population. In addition, we must also configure some genetic algorithm specific hyperparameters that determine how our population changes over time and the running time of our algorithm (see Table 2).

We also used genetic algorithms for feature selection. Prior research has shown that genetic algorithms are a good choice for efficient feature selection [8]. For our model, each time series datapoint and each statistical feature can either be selected or not for each individual. This is useful in finding which statistical features are relevant to our classification of gender and which parts of the time series contain information that can help distinguish the gender of the subject. Thus, a third set of models was trained that used both genetic hyperparameter search and genetic feature selection.

Table 2: The hyperparameters values used for our genetic algorithm.

Genetic Hyperparameters	Value
Population Size	20
Mutation Probability	.1
Elitism ratio	.05
Crossover Probability	.5
Parents Portion	.3
Crossover Type	Uniform
Max iterations	10

3.6 Bidirectional Architecture and parameter tuning

The second model proposed will have a similar architecture but a different training scheme: a bidirectional neural network (BDNN). This type of network is trained in cycles of forwards and backward passes. The forwards training cycle will take the inputs into the model and use the traditional backpropagation algorithm to compute the loss in the outputs. The backwards training cycle will do the opposite, taking the previous outputs of the model, and using backpropagation to compute the loss in the original inputs. This type of model is believed to be a "key factor in reducing network generalisation error" which is useful given our small dataset and susceptibility to problems with generalization [9].

One important feature of BDNNs is that they need to be invertible, or that the inputs and outputs of a problem map one to one. Classification is clearly not invertible so we need some kind of workaround to build a BDNN for the problem at hand in Nejad and Gedeon, they give two examples of how to make the outputs of a classifier invertible. The first involves creating an invertible function for classifications of final marks based on some input features such as grades during the semester. This used many statistical assumptions, such as monotonically increasing functions that map attributes to the classifications, that do not apply to our problem.

The second case inverted digit classification by encoding the extra information from the inputs in an additional output node. In our case, we can take advantage of this approach to create a pseudo invertible mapping by adding additional output nodes to our network and only using one node to calculate loss when predicting gender. From this, we can fully describe the BDNN architecture we will use for our model as seen in Fig. 3. This is a sample BDNN, but our BDNNs have the same architecture as our composite feed forward network, but with one fewer hidden layer.

Much like the feed forward neural network we use for our first model, our BDNN can leverage LOOCV, random search to tune hyperparameters and dropout layers. In addition, we also need to set a hyperparameter to control epochs trained in the backwards direction. Traditionally, BDNNs are trained for some fixed number of epochs in the forwards direction, then reversed and trained until the loss is below either some relative threshold or below the

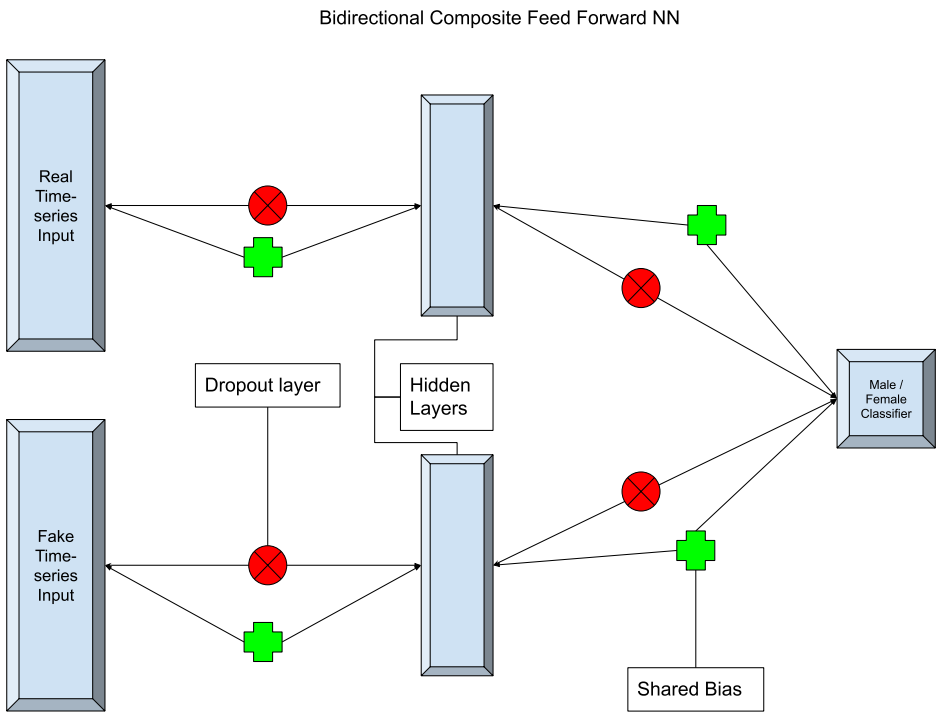

Fig. 3: Our bidirectional neural network architecture. Note the shared biases and lack of a second hidden layer.

forwards network loss. If the network is unable to reduce loss fast enough, the training resumes in the forward direction after some fixed number of epochs. This number and the total number of cycles are additional hyperparameters that we have tuned using random search and are also shown in Table 1.

The final implementation detail for BDNNs is that the forwards and backwards networks share weights and biases. When implementing our BDNN, it was necessary that we copied the weights from the forward network to the backwards network after changing directions, and vice versa when flipping from the backwards network to the forwards one. There is a small subtlety here in that the bias must be copied as well, but that the bias for the last layer in the forward and backward network are distinct. This is due to the fact that the last layer in the forward network corresponds to the inputs for the backwards network and vice versa, meaning that there is no associated bias.

4 Results

The actual results from our first models, the feed forward neural networks, are lackluster. Fig. 4 and Fig. 5 show that although our model is able to have reasonable accuracy in our training and validation sets, it only accurately classifies one out of two of our training examples for all but two network architectures. Not only that, but even if the model could classify both correctly, we have no way of knowing that it actually performs well in out of sample individuals given that we only have two cases in the test set.

Relative to our first model, the results from the BDNN model are similar. However, further analysis of the loss during training tells a different story. While the forward network is able to converge, the loss for the backwards network remains high and it is unable to converge to any reasonable optima for the network parameters. This may be due to the difficulty in turning this classification problem into an invertible problem.

Finally, let us compare the results from each of the network architectures. For the feed forward networks, training solely on statistical data gave us worse results. Not only did training on all inputs (time series and provided statistical features) result in better validation set distribution, but it also resulted in better test performance. Although this is not statistically significant, it warrants future research on a larger dataset. The BDNN architectures on the other hand resulted in much more stable validation results. Both networks resulted in above average accuracy and stable classification results. Note that for the BDNN that used both time series and statistical features, the validation results were identical for all results. Upon further investigation, it appears that this architecture is converging to the same set of weights consistently. This seems a bit odd given the random search of hyperparameters so future work could involve investigation into this phenomenon and if it is a property of BDNNs or a failure in our implementation.

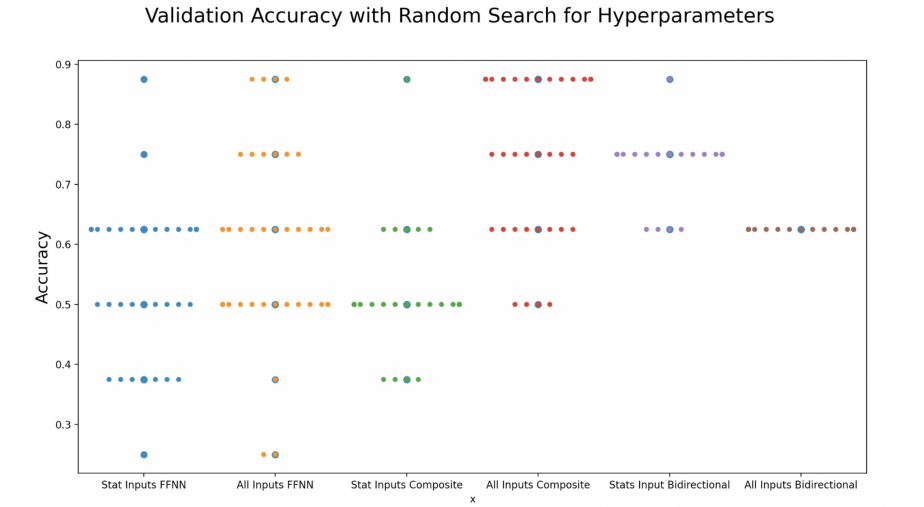

Fig. 4: Our validation results for all neural networks with random search hyperparameter optimization.

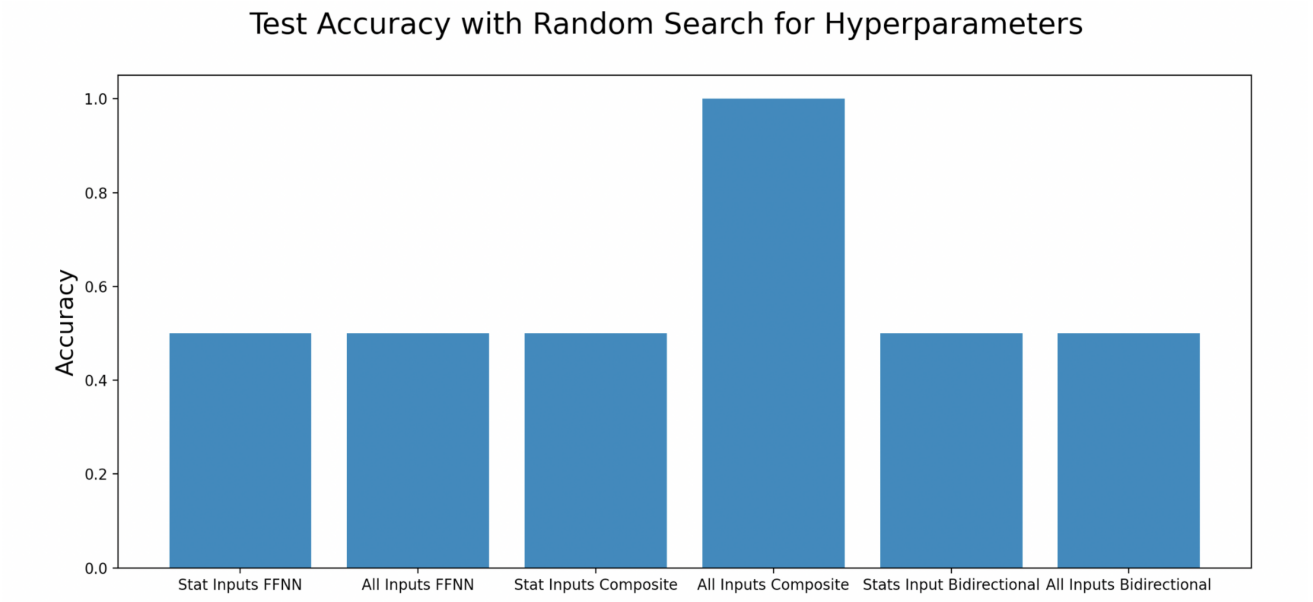

Fig. 5: Our test results for all neural networks with random search hyperparameter optimization.

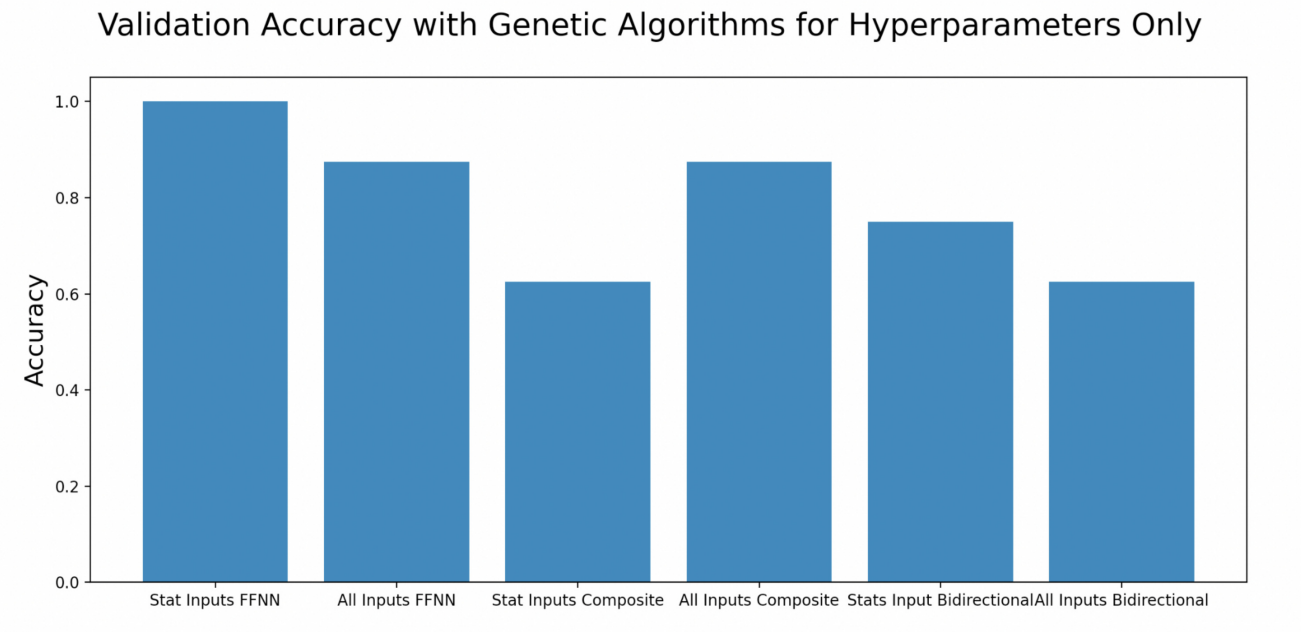

Fig. 6: Our validation results for all neural networks with genetic algorithm hyperparameter optimization.

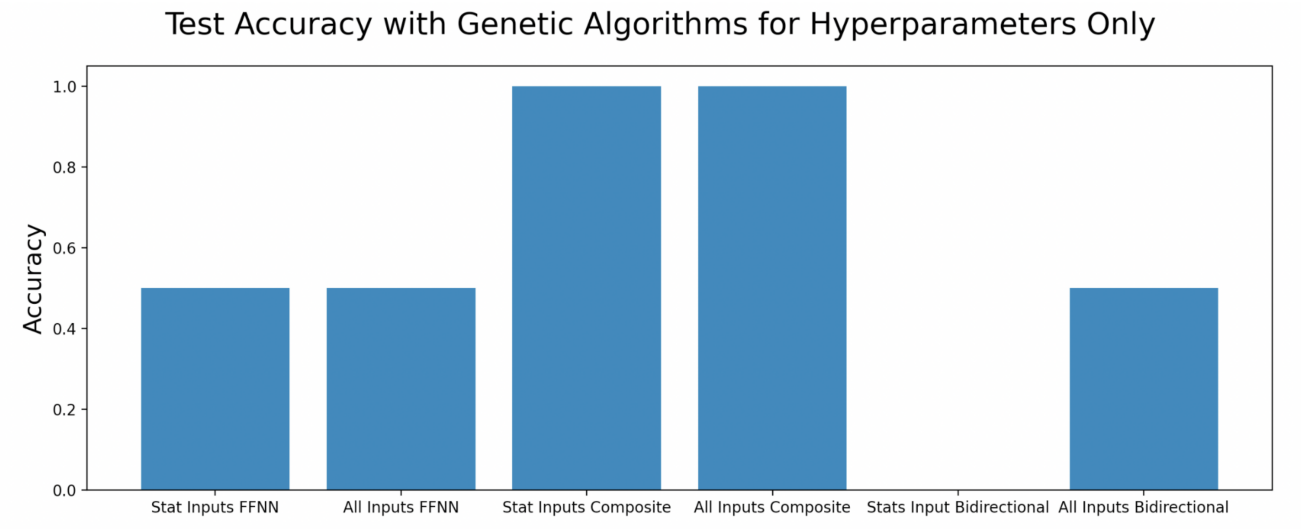

Fig. 7: Our test results for all neural networks with genetic algorithm hyperparameter optimization.

Now we will discuss the results of the same sets of neural network architectures that were optimized using genetic algorithms for feature selection and hyperparameters. Fig. 6 and Fig. 7 display the validation and test results for our models trained with hyperparameters optimized by a genetic algorithm. Compared to the validation results from the random search hyperparameter models, there is no significant improvement in the validation set and in fact, there are some models in the test set (namely, the model trained on only statistical features using a bidirectional NN architecture) that perform worse for genetic algorithms. This may be due to running the genetic algorithm for a low number of iterations due to time constraints, essentially rendering it no different than the random search.

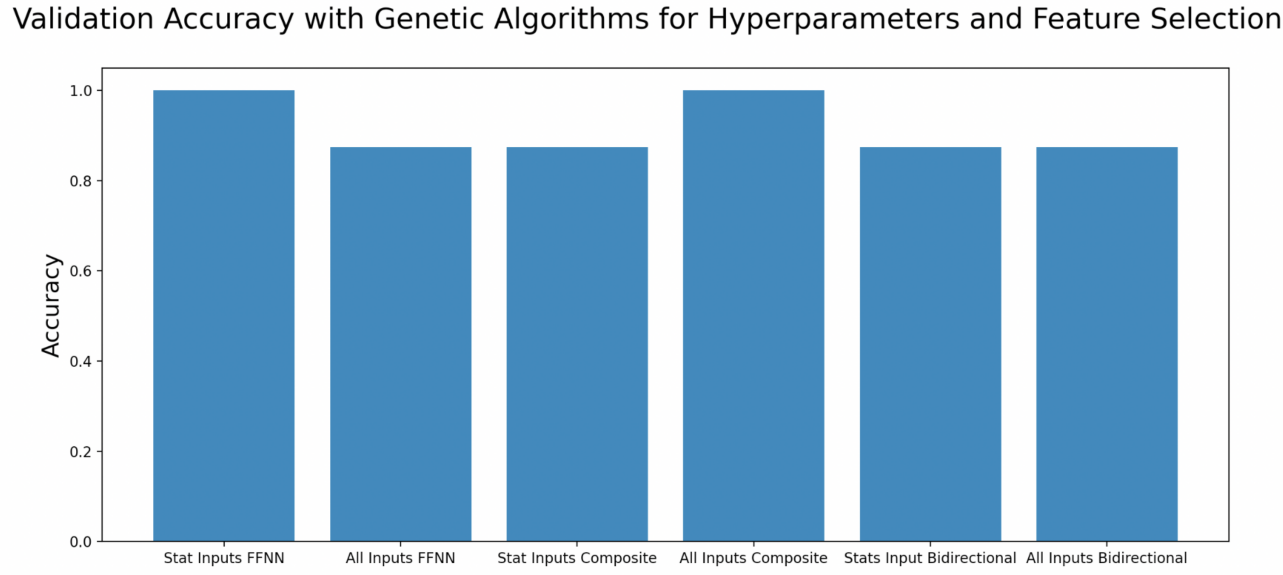

Fig. 8: Our validation results for all neural networks with genetic algorithm hyperparameter optimization and feature selection.

Table 3: The hyperparameters for our final model.

Neural Network Hyperparameters	Lower Bound
Learning Rate	3.134×10^{-3}
Dropout Rate	.275
First Hidden Layer Size	5
Second Hidden Layer Size	4

Our final set of models was trained using genetic algorithms for both feature selection and hyperparameter tuning. The results across the board were improved for the validation set and were similar for the test set (see Fig. 8 and Fig. 9). This is a good result, but future investigation on a larger dataset would help us determine if the result is solely due to overfitting on feature selection. Given the small sample size, it may be possible to select features that result in good accuracy for this dataset, but do not generalize.

The remainder of the analysis will be on the results of our best model, the composite feed forward neural network with time series and statistical inputs with both genetic algorithms for hyperparameter tuning and feature selection. Our results showed that there was no pattern in time series feature selection (there was a uniform distribution of time series data points selected) but only a handful of statistical features were selected. These are listed below along with the hyperparameters that were used to train the final model in Table 3 and Table 4.

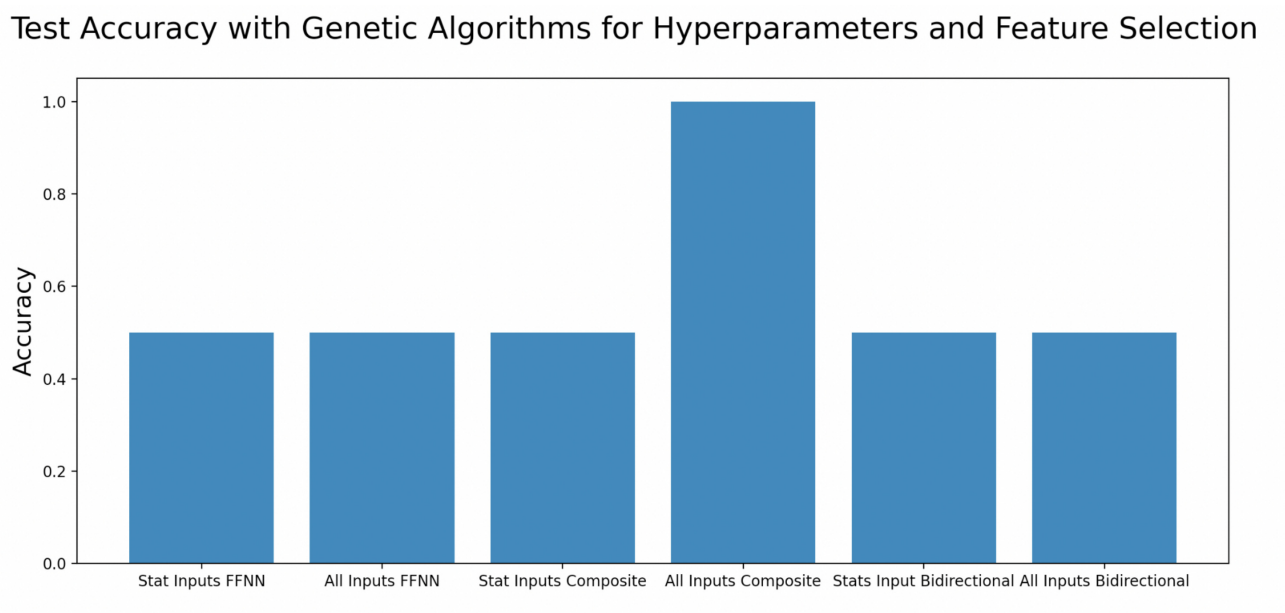

Fig. 9: Our test results for all neural networks with genetic algorithm hyperparameter optimization and feature selection.

Table 4: The statistical features selected for our final model.

Statistical Feature
Mean
Min
Sum
Root Mean Squared
Hjorth Mobility
Hurst Exponent
Sample Entropy
Approximate Entropy

5 Future Work

Although the evaluations and statistical results of both our networks are lackluster, this is almost entirely due to the limitations of the dataset. With a larger training set and more data from individuals for a larger test set, the approach considered here could be evaluated more thoroughly. In addition, this sample only included Asian individuals and should be expanded to include individuals of other races as well to determine if the network can generalize to all ethnicities.

There are several areas for future work that should be mentioned. First, the input data is primarily time series which is ideal for training recurrent neural networks. For these networks, the bidirectional approach we have taken is particularly pertinent as bidirectional networks have been common among Long Short Term Memory architectures for some time [10].

Also, research into why particular statistical features were selected by our feature selection could reveal relationships between pupillary responses and the gender of the observer. Investigation into if the entropy, Hjorth parameter or Hurst exponent of the timeseries of pupillary dilations are more information rich features than others could be a source of future research.

References

1. Hossain MZ, Gedeon T, Sankaranarayana R, Apthorp D, Dawel A. Pupillary responses of Asian observers in discriminating real from fake smiles: A preliminary study. Measuring Behavior. 2016. pp. 170–176.
2. Du K-L, Swamy MNS. Neural Networks and Statistical Learning. Springer Nature; 2019.
3. Cunningham P. Dimension Reduction. Machine Learning Techniques for Multimedia. Springer, Berlin, Heidelberg; 2008. pp. 91–112.
4. Kearns M, Ron D. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. Neural Comput. 1999;11: 1427–1453.
5. Zaremba W, Sutskever I, Vinyals O. Recurrent Neural Network Regularization. 2014. Available: http://arxiv.org/abs/1409.2329
6. Bergstra J, Bengio Y. Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research. 2/2012 [cited 12 Apr 2021]. Available: https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a
7. Wicaksono AS, Supianto AA. Hyper Parameter Optimization using Genetic Algorithm on Machine Learning Methods for Online News Popularity Prediction. International Journal of Advanced Computer Science and Application. 2018;9. Available: https://thesai.org/Downloads/Volume9No12/Paper_38-Hyper_Parameter_Optimization.pdf
8. Tan F, Fu X, Zhang Y, Bourgeois AG. A genetic algorithm-based method for feature subset selection. Soft Computing. 2007. pp. 111–120. doi:10.1007/s00500-007-0193-8
9. Bidirectional neural networks and class prototypes. [cited 12 Apr 2021] Available: https://ieeexplore.ieee.org/abstract/document/487348
10. Decomposition-based hybrid wind speed forecasting model using deep bidirectional LSTM networks. Energy Convers Manage. 2021;234: 113944.