Applications of Neural Networks to Facial Interpretation.

Group 14

J Mair, O Hussein, B Foong, H Li and H Yuan

I. INTRODUCTION

II. METHOD

A. Preprocessing

We processed each feature set into a 2D array in which each column represents a single sample of features. In the case of the binary and regression data this involved reshaping the two spacial dimensions of each facial point into a single dimension.

For the multi-class label set we needed to convert each category label (integer values of 1 to 6) into "One-Hot Encoding"[1], due to it allowing a better description of an error function. The error function we can use therefore is cross-entropy which allows us to qualify definitive predictions as better, which is expected to increase the generalisation ability of a network. The original integer makes the output orderable orderable[1], which is unexpressive for classification problems.

From the regression data set we extracted the 6th label from each sample which represented the yaw of the head.

Finally we produced a random permutation of each data and label set pair to allow random train/test split of the data in our experiments. This was done using a seeded random number generator so that the results were repeatable.

B. K-Nearest Neighbour Tests

As a baseline, we applied variants of the K-Nearest Neighbour (KNN) algorithm to each different data set. On the two classification problems, we cross validated the KNN algorithm with 4 folds and plotted a confusion matrix as seen in Figure 1 and Figure 2. These gained accuracies of (81±6)% and (54±4)% for the binary and mutli-class problem respectively, averaged across a 4 fold cross-validation set.

For the regression problem we slightly altered the KNN algorithm so that it would take the mean of the k nearest labels instead of the mode, due to the continuous nature of the output. This algorithm was used with k=3 and 4 fold cross-validation to obtain root mean square error as 1.8 ± 0.3.

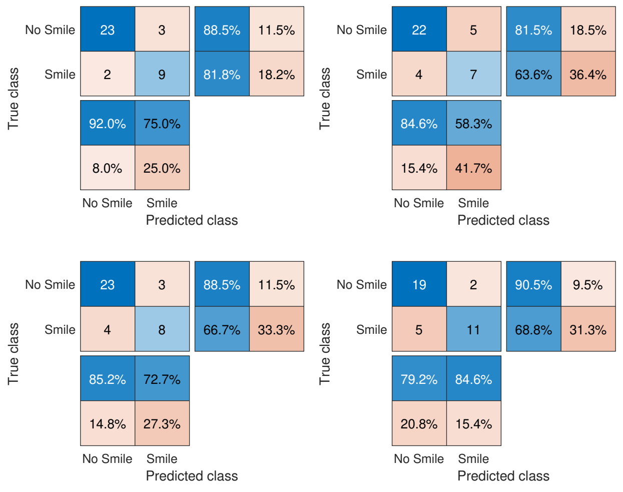

FIG. 1. Classification results for the KNN algorithm (k=3) on the binary problem with 4 fold cross-validation. This algorithm got a mean F1 score of 0.86±0.03 for class 0 (not smile) and 0.71±0.08.

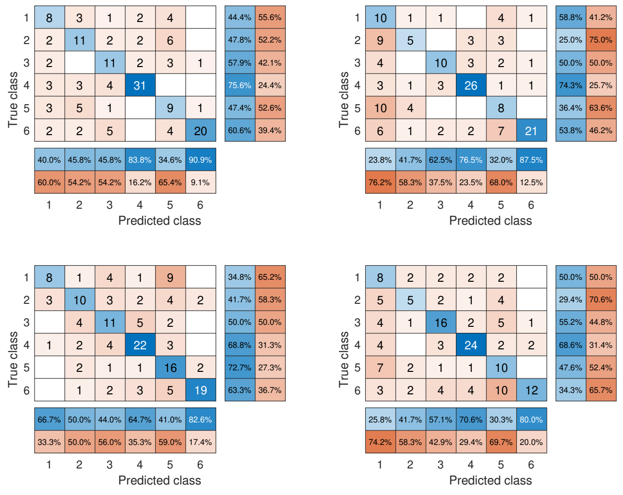

FIG. 2. Classification results for the KNN algorithm (k=5) on the multiclass problem with 4 fold cross-validation. This algorithm got mean F1 scores of 0.39, 0.40, 0.52, 0.73, 0.41 and 0.65 for the classes 1 to 6 respectively.

C. Neural Networks

1. *Binary Classification*

Upon inspecting this data set we noticed that the distribution of classes was weighted towards the "No Smile" class. We tried to avoid biasing the network and as a

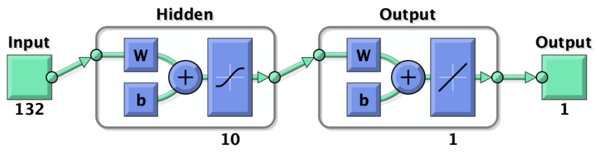

FIG. 3. Network topology for the binary neural network. This was trained using Bayesian regularization backpropagation[2], epochs set to 100 and the Marquardt adjustment parameter set to 0.005.

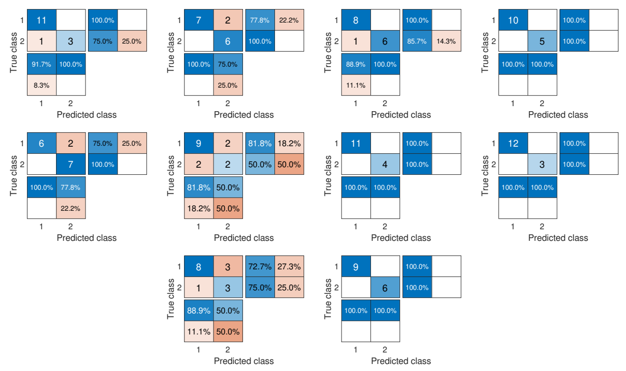

FIG. 4. The confusion matrices for each different cross validation upon training the network in Figure 3.

result we choose the network topology and hyper parameters that resulted in better F1 scores, since the accuracy would not be enough.

In training we used the mean square error as a performance measure and so we reasoned that the network would make predicts close to 0 or 1. To make a decision we used the following formula:

$$Y_i = \lfloor \max(0, \min(y_i, 1)) \rceil \tag{1}$$

, where Y_i is the actual prediction and y_i is the output from the network.

Upon training we decided on a simple network topology and hyper parameters, which is shown in Figure 3, since our data set was extremely small. We tried to minimise the number of intrinsic parameter to avoid overfitting; this included adding early stopping mechanisms by changing the number of epochs and validation checks to a smaller number.

Training this network resulted in a mean F1 score of 0.92 for class 0 (not smile) and 0.86 for class 1 (smile) and a mean accuracy of (91±2)% across the 10 fold cross-validation sets.

2. *Multi-class Classification*

After receiving the preprocessed data from earlier, we experimented with different training mechanisms. At first we used the default *feedforwardnet* supplied in the MATLAB toolbox[3], however we compared it to using *patternnet* instead and found that this performed better on F1 scores from the confusion matrix.

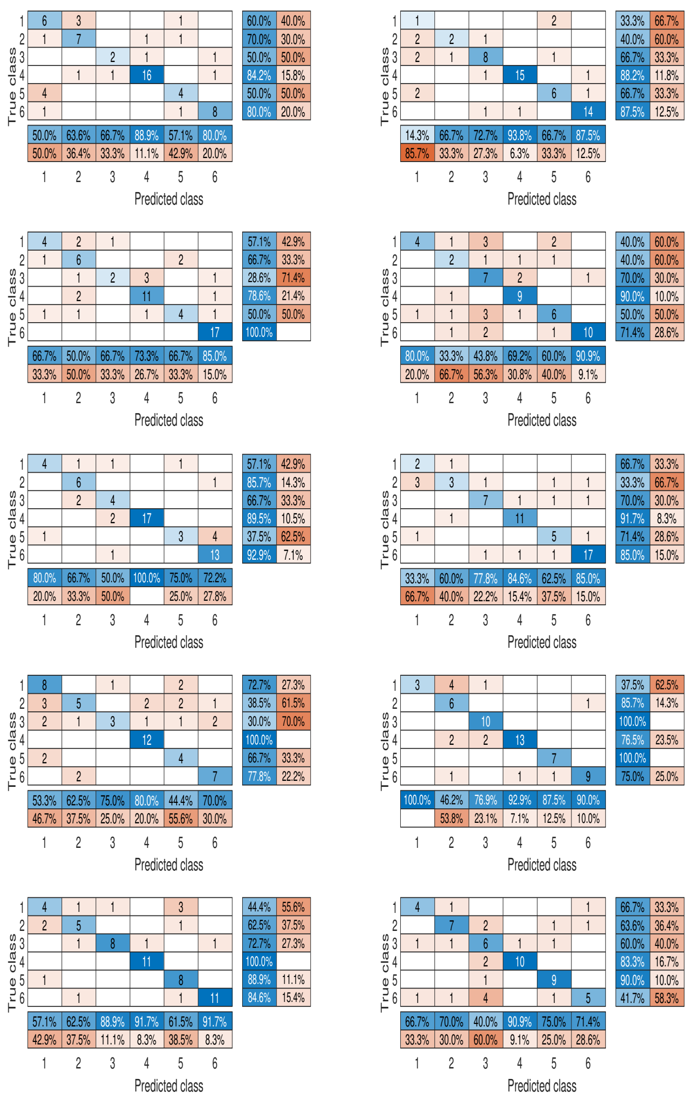

FIG. 5. The confusion matrices yielded from each of the 10 cross validation sets.

We attempted to use meta optimisation on this problem but found that the data set was far too small and any major meta optimisation would result in over fitting and a biased value of the mean. Instead we stopped our tests early and picked a set of parameters that achieved an average classification accuracy of (70±6)%. This network architecture is shown in Figure 6.

Our results yielded a mean confusion matrix as shown in Figure 5 and F1 result over all classes and cross validation set of 0.56, 0.58, 0.62, 0.86, 0.72 and 0.81 for the classes 1 to 6 respectively. This yielded an accuracy of (73±6)%.

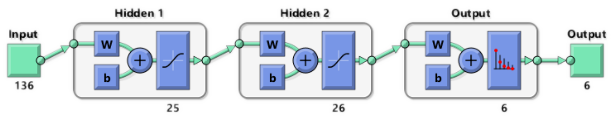

FIG. 6. Network topology for the multi-class neural network. This was trained using Scaled conjugate gradient backpropagation[4], with sigma set to 0.00036915, lambda set to 0.00025801, epochs set to 500 and max fail set to 10.

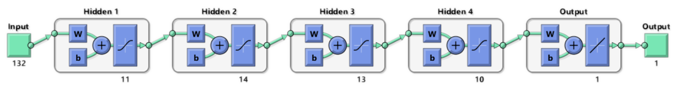

FIG. 7. Network topology for the regression neural network. This was trained using Levenberg-Marquardt backpropagation[6] with epochs set to 500, max fail at 5, mu at 69.242 and minimum gradient set to 2.3122. All other settings were at their default.

3. Regression

For our final set we tried a few network structures. Despite some success with the two classification models, our attempts to manually adjust our hyper-parameters to achieve a MSE below the KNN results were futile. We therefore used a hyper heuristic to tune our hyper parameters. Our aim was to minimise the average MSE under cross-validation. If we were to simply apply this to our entire data set we would be at risk of over fitting to our sample data. To avoid this we randomly split the data set having 40% to try and find more optimal hyper-parameters and leaving 60% to do a final cross validation check on the provided settings.

The hyper heuristic used was the bayesopt[5] function. This function was allowed to alter the topology of the network (the number of neurons in each of the 4 hidden layers) while also changing two of the hyper parameters of the learning function[6]. The hyper heuristic selected the topology and hyper parameters shown in Figure 7.

These parameters were used structure and train a neural network to which got an average root mean square error of 1.8±0.3.

III. CONCLUSIONS

[1] Amanda Casari Alice Zheng. *Feature Engineering for Machine Learning*. O'Reilly, 2018.
[2] Bayesian regularization backpropagation. `https://uk.mathworks.com/help/deeplearning/ref/trainbr.html`. Accessed: May 14, 2020.
[3] Matlab deep learning toolbox. `https://uk.mathworks.com/products/deep-learning.html`. Accessed: May 14, 2020.
[4] Scaled conjugate gradient backpropagation. `https://uk.mathworks.com/help/deeplearning/ref/trainscg.html`. Accessed: May 14, 2020.
[5] Bayesian optimization. `https://uk.mathworks.com/help/stats/bayesopt.html`. Accessed: May 14, 2020.
[6] Levenberg-marquardt backpropagation. `https://uk.mathworks.com/help/deeplearning/ref/trainlm.html`. Accessed: May 14, 2020.