Sahba Mobini Farahani (A13510656)
Saidel Halol (A14024702)
Rajandeep Kaur (A13736425)
Anona Gupta (A13606136)
Nicole Chang (A14780905)
Group 63: Saran Wrap

118B Final Paper: Age Group Detection By Facial Image

Introduction and Motivation

In this project, our group decided to use FaceTracer Database, which is a large collection of real-world face images, collected from the internet from the University of Columbia in order to construct a support vector machine (SVM) and neural network which would correctly classify the age of a person in a given picture. Initially starting with a different data set, our group's curiosity was to see the role that facial features played when trying to categorize important features such as age or age group. We found that the FaceTracer Database was the most precise dataset and had all of the information we were looking for, such as a clear age group, facial angles, and relationships between facial features. Face classification can be an especially useful tool for social media companies or anyone looking to detect certain aspects of a person through their face– aspects such as age, emotion or facial hair. Some uses of facial classification are preventing crimes like shoplifting or cheating in casinos, security in unlocking phones, and targeted advertising. Its potential societal uses drove our group to learn more about what algorithms gave the best results and better understand the implementation behind those algorithms.

Related Work

Similarly to our project, Mohammad Mahdi Dehshibi and Azam Bestanfard created an algorithm using neural networks to classify age based on facial images. Using facial feature ratios and wrinkle densities, they were able to correctly identify age groups based on faces with an accuracy of 86.64%. Another experiment that classified age based on facial images was done by Yi-Wen Chen *et al*. using SVM. By building an active appearance model, they extracted the features that would be used to differentiate between the age groups. They then compared the age groups with the largest difference, the elderly and children, using hierarchical SVM and then using those results, they reclassified the adult group. This method yielded an accuracy of 87.8%. We saw another example of SVM being implemented with facial data to test facial recognition from P. Jonathan Phillips of the National Institute of Standards and Technology. Their resulting data concluded that, with their 'difference space' SVM (i.e. drawing a line between their designated two classes of faces), that support vector machines make better use from facial data

than “baseline PCA algorithms”. This work split the data on the line between the differences in facial image of a single individual and the differences in facial image of different individuals. They found that their error rate using SVM was almost half of that through PCA methods, 7% versus 13%.

Methods

The data we used from the FaceTracer Database was contained in two files, ‘facelabels’ and ‘facestats’. The ‘facelabels’ file contains data about select faces that had been assigned attribute labels. The different faces have unique face ids. The ‘facestats’ file contains statistics about each face in the database. These statistics include width and height of the face in pixels, pose angles and fiducial points. Similar to the ‘facelabels’ file, each face has a unique face id.

In order to clean our data, we first dropped the column in our ‘facelabels’ dataset that indicated which attribute the values in the ‘label’ column were a subset of, since the only attribute we needed was age group, making this column unnecessary. We then removed all rows that didn’t contain an age group label by keeping only the first 1000 rows of the dataset, as these were the only rows with the age group attribute. Since we needed to merge our ‘facelabels’ and ‘facestats’ datasets based on the face id, we set the ‘face_id’ column to be the index for both datasets. When merged, this would only keep the data from ‘facestats’ that applied to the faces we had age group labels for and prevent from having missing data. To keep our data together in one dataset and allow for our techniques to be done to the data, we merged the two datasets.

For the next step, we utilized one of the more popular machine learning algorithms to classify our data. SVM is a linear model that was originally made for binary classifications but was later expanded for multiclass classifications as well. On a basic level, the algorithm is used to classify data by creating a line or hyperplane that separates the classes. A couple ways to tackle multiclass classifications would be to use one-vs-one (OVO) or one-vs-rest (OVR) strategies. With the OVR approach, we would have to train 5 classifiers since we have 5 different classes (baby, child, youth, middle-aged, and senior). This would include training baby vs. rest, child vs. rest, youth vs. rest, middle-aged vs. rest, and senior vs. rest. The OVO approach is based on training one classifier on each pair of classes resulting in a total of [n_classes(n_classes - 1)/2] classes, which in our case would be 10 classes. The difference between the two approaches lies within the classification phase where OVR predicts the probability and chooses the class with the highest probability and OVO chooses the class that’s been predicted the most out of the 10 classes. However, given the complication of combining votes for both of these approaches and being more computationally expensive, we decided to go with a more simplistic approach to see if our trained models could accurately predict between easy and hard cases. An easy case would be classifying between baby and senior and a hard case would be classifying between baby and child.

For the easy case, data was taken from the classes labeled baby and senior then split into 80% training and 20% testing data for both features and classes. Before running the training data through the classifiers, all features were scaled to a mean of 0 and a variance of 1 using standard scaler from sklearn.preprocessing. Scaling data avoids features with greater numeric ranges to overpower the smaller numeric data. One of the perks of using SVM is the use of different kernels. Since we used linear and Gaussian Radial Basis Function (RBF) kernels, we avoid having numerical problems by scaling our data.

After splitting and scaling the data, we ran a grid search cross validation to find the best fitting hyperparameters to train our classifier with. The parameters used for a linear SVM was a regularization parameter (C) of 10e-4, 10e-3, 10e-2, 1, 10e1, 10e2, cross validation of 10, and the number of job to run in parallel as -1. RBF had the same parameters except an additional gamma parameter was added (1e3, 1e2, 1, 1e-2, 1e-3, 1e-5). We ran both kernels separately and used the optimal parameters to train each classifier to fit our data. Lastly, we computed accuracy, confusion matrix, and classification matrix for both training and testing data to see how well our models predicted age.

https://github.com/changnicole/118B-Final-Project/blob/master/118B%20Final.ipynb

```
mlp = MLPClassifier(solver='lbfgs')
mlp.hidden_layer_sizes = (8, 8)
```

```
mlp.fit(train_img, train_lbl)
print("Training set score: %f" % mlp.score(train_img, train_lbl))
print("Test set score: %f" % mlp.score(test_img, test_lbl))
```

```
Training set score: 0.166861
Test set score: 0.118881
```

```
mlp = MLPClassifier(solver='lbfgs')
mlp.hidden_layer_sizes = (64, 64)
```

```
mlp.fit(train_img, train_lbl)
print("Training set score: %f" % mlp.score(train_img, train_lbl))
print("Test set score: %f" % mlp.score(test_img, test_lbl))
```

```
Training set score: 0.863477
Test set score: 0.237762
```

```
mlp = MLPClassifier(solver='lbfgs')
mlp.hidden_layer_sizes = (128, 128)
```

```
mlp.fit(train_img, train_lbl)
print("Training set score: %f" % mlp.score(train_img, train_lbl))
print("Test set score: %f" % mlp.score(test_img, test_lbl))
```

```
Training set score: 0.919487
Test set score: 0.223776
```

Another classification method we tried to implement was neural networks. We tried two layer neural networks, as suggested by the TA's. Using the same data as in the other methods, first the data was scaled, and **labels one-hot encoded**. The dataset was split into 80% training and 20% testing. Then we applied a simple deep learning model with sklearn's MultiLayered Perceptron Classifier. The default optimization function is 'Adam', but since our training data was less than 1000 samples, we used 'LBFGS', the suggested optimization function for smaller training data in sklearn documentation. Using the MLP Classifier with smaller number of nodes per layer, accuracy was very low and with larger number of nodes, the model **overfit**. After inquiring on Piazza on how to address overfitting in neural networks, we decided to switch

to a Sequential model, using keras and tensorflow libraries, specifically to add **dropout** layers, and also better finetuning. The model had an initial layer with rectified linear activation(**Relu**) function. Then it had two hidden layers. Finally, since our problem is multiclass, we applied a **softmax** function and then fit using a **categorical cross entropy** loss function. Using the sequential model, a dropout layer was added after each hidden layer of the network, as well as a validation split, with 80% train and 20% validation, within each epoch. The dropout rate was 0.5. The optimizer used was **Adam**, and we experimented with a few different **hidden layer sizes** as well as number of **epochs**, as shown in the results section.

https://github.com/anonarai/Cogs118B_FinalProject/blob/master/FaceTracer%202%20Layer%20Neural%20Net.ipynb

Results

On the easy classification with baby versus senior, linear SVM did far better than the RBF kernel. Linear SVM resulted with a training accuracy of 85.5%, validation accuracy of 82.5%, and testing accuracy of 78.8% with the best C parameter being 100 whereas RBF produced results of 100%, 72.5%, and 71.3% respectively with the best C parameter being 1000 and gamma being 0.01. Given the higher testing accuracy with the linear SVM compared to the RBF, we can infer that the linear SVM is better at classifying between baby and senior. Another thing to note is that an RBF kernel is much easier to overfit since it's a complex model compared to linear SVM which makes sense since the training accuracy is a full 100% and drops 30% when being tested. Both kernels' performance on the hard classification resulted in a similar pattern where linear SVM did better than RBF. However the results for the accuracies were lower than those in the easy classification. The best C parameter that we ran for linear SVM was 1 and it led to a training accuracy of 74%, validation accuracy of 68.8%, and testing accuracy of 71.3%. RBF, on the other hand, had a training accuracy of 100%, validation accuracy of 68.1%, and testing accuracy of 68.8% with the best parameters being C = 1000 and gamma = 0.001. When solely comparing the testing accuracy and how well each model did on both the easy and hard classifications, it was relatively easy for the models to be able to classify baby faces versus senior faces yet difficult to classify baby faces compared to child faces.

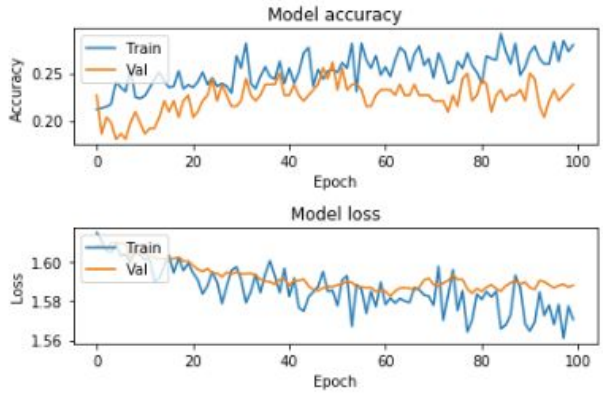
When comparing the confusion metrics of linear SVM and RBF for the easy classification, we can see that the precision values of linear SVM, 79% for predicting baby faces and 78% for predicting senior faces, are higher than those of RBF, 72% for predicting baby faces and 71% for predicting senior faces. This indicates that out of the faces that it predicted to be a baby, the linear SVM kernel was correct more often than the RBF kernel, and the same applies to the predictions for seniors. Additionally, linear SVM gives higher recall values at 78% for predicting baby faces and 80% for predicting senior faces compared to RBF's 70% for predicting baby faces and 72% for predicting senior faces. This means that linear SVM provided a higher

ratio of correct predictions of baby face classifications out of the total baby faces being classified, and the same applies to the predictions for seniors. As for F1 scores, linear SVM had scores of 78% for predicting baby faces and 79% for predicting senior faces whereas RBF had scores of 71% for predicting baby faces and 72% for predicting senior faces, showing that linear SVM incorrectly classified faces less than RBF did. Based on these results, we can infer that linear SVM is better at classifying baby faces from senior faces, and vice versa, than RBF.

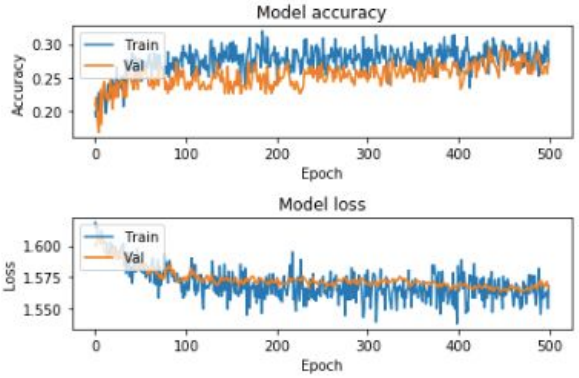
The confusion metrics of the hard classification display a similar pattern to those of the easy classification. The values from the linear SVM kernel were higher than their respective alternatives from the RBF kernel. In comparison to the easy classification though, the hard classification resulted in lower values in its confusion matrix. Both kernels had a drop in the ratio of true positives (e.g. correctly predicting a baby face) to overall positives predicted (e.g. all predictions of a baby face) and true positives to actual positives (e.g. all baby faces), as well as in increase in false positives (e.g. incorrectly predicting a baby face) and false negatives (e.g. incorrectly predicting that a face is not a baby face). From these results, we can conclude that both kernels are better at distinguishing between babies and seniors than babies and children. Overall, the highest values came from using linear SVM to predict whether a face is that of a baby's or a senior's.

When implementing the neural networks, first we learned that with a higher number of nodes in hidden layer the accuracy did increase, but there was a limit. While this may not be the case for all data, it was for ours. We also learned if the number of nodes was too small, then overfitting was not a problem but accuracy was low all around. After switching to the keras library, applying dropout did help significantly with overfitting, though after a certain point did not lead to any improvement in test accuracy. Also adding dropout with a smaller rate after the input layer, did seem to help with loss and accuracy scores. However a keeping that same small rate, 0.2, in between hidden layers was less effective in preventing overfitting than a rate of 0.5. We also noticed that the accuracy seemed to hover around 35-37% after the number of nodes > 64 and 100 < epochs < 500. Increasing the number of epochs from 50 did lead to changes, however the model improved very slowly, and after 500 epochs, improvements in loss and accuracy began to decrease again.

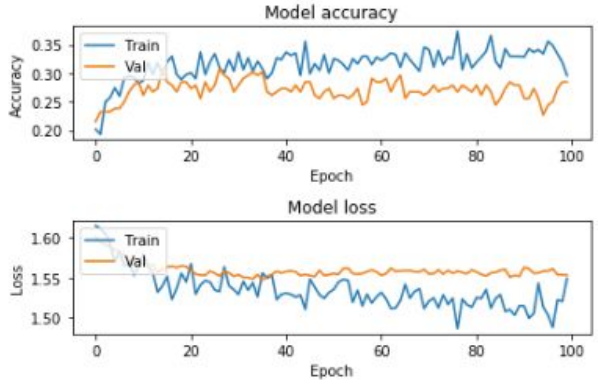
optimizer: Adam, epochs: 100, layer sizes: 8
Test loss score: 1.580509079086197
Test accuracy: 0.28671327



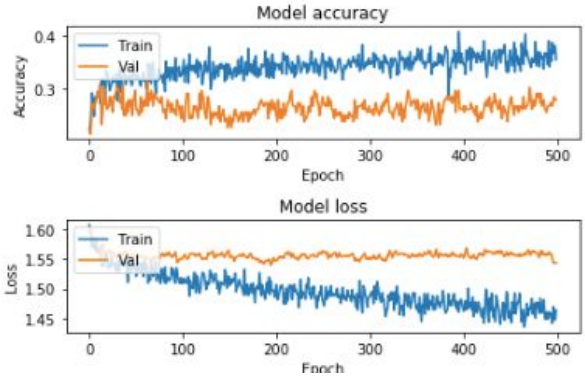
optimizer: Adam, epochs: 500, layer sizes: 8
Test loss score: 1.5552121267452106
Test accuracy: 0.30769232



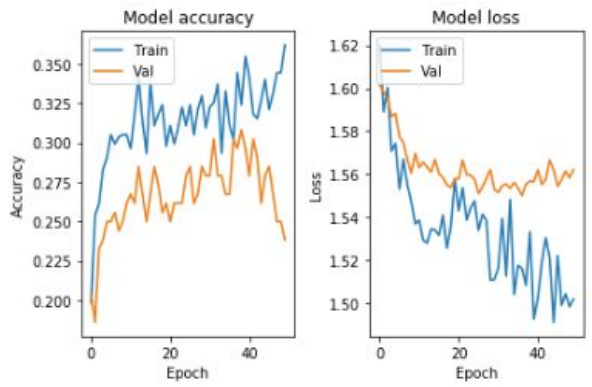
optimizer: Adam, epochs: 100, layer sizes: 64
Test loss score: 1.5165602635670374
Test accuracy: 0.34265736



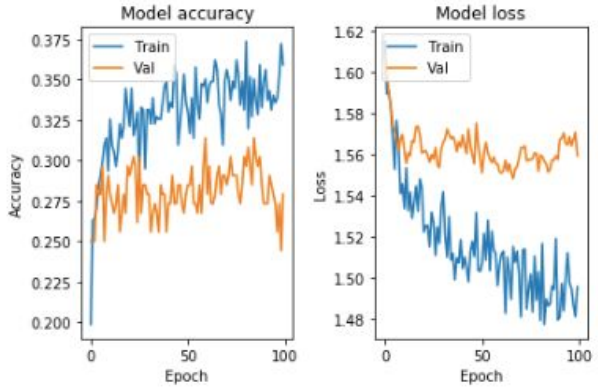
optimizer: Adam, epochs: 500, layer sizes: 64
Test loss score: 1.5321731442338102
Test accuracy: 0.37062937



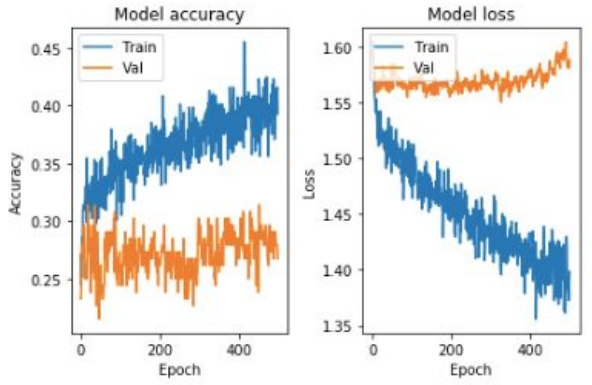
```
optimizer: Adam, epochs: 50, layer sizes: 128
Test loss score: 1.5205936790346266
Test accuracy: 0.35664335
```



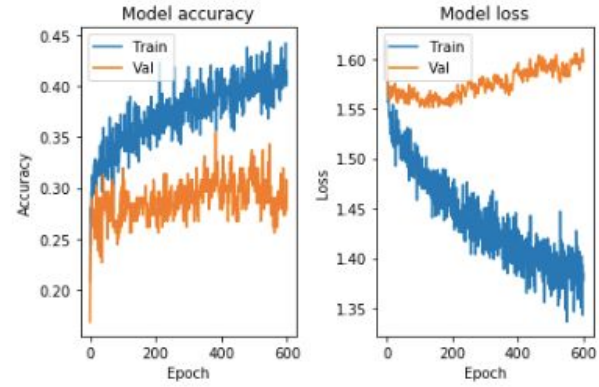

```
optimizer: Adam, epochs: 100, layer sizes: 128
Test loss score: 1.5277309826203993
Test accuracy: 0.36363637
```




```
optimizer: Adam, epochs: 500, layer sizes: 128
Test loss score: 1.5920372259366762
Test accuracy: 0.37762237
```




```
optimizer: Adam, epochs: 600, layer sizes: 128
Test loss score: 1.633004269399843
Test accuracy: 0.30769232
```



Discussion

We learned a lot about fine tuning neural networks and the different aspects involved like optimizers, type of loss function, epochs and more. We also learned more about multi classification problems, and how many extra steps were needed. Of course, the original research paper that our data came from used a custom Attribute-Tuned Global SVM model to accurately classify the data, and not neural networks. It could be interesting to do even more research into when SVMs work better for multi classification problems versus neural nets. Also, it is possible that ~800 data entries was not sufficient to train a neural network model well. In addition we learned there are so many different features of a neural network that we simply did not have time to dig deeper into. While here we focused on the size of the layers and the number of epochs, it would also be interesting to test the effects of things like batch size, validation percentages, or compare across optimizers. Also it is possible that the biases were not optimal and so learning how to influence and optimize those may have helped. A main issue could also be our data cleaning, it's possible that there is some better way to standardize the data, which would influence everything downstream. Another direction to look into would be using PCA on the

data first to see if only certain features are very important, and then simply classify with those using SVM. Finally, most interesting was that while the loss and accuracy did change a bit with different parameters in the neural net, they did not vary *that much,* which leads us to ask how large or small improvements must be for increased time and computational capacity to still be worthwhile, at least in deep learning. It also shows how much the data affects the results, regardless of the complexity of the classifier. Given more time, on top of utilizing PCA with SVM, we would have loved to explore the many different combinations of classifications with OVO and OVR with other classifiers and not just SVM along with computing the voting process to determine the final class.

References

Chen, Y., Han, M., Song, K., & Ho, Y. (2010). Image-based age-group classification design using facial features. *2010 International Conference on System Science and Engineering*, 548-552.

Dehshibi, M. M. & Bastanfard, A. (2010). A new algorithm for age recognition from facial images. *Signal Processing*, *90*(8), 2431-2444.

Philips, Jonathan P. (1999) Support Vector Machines Applied to Face Recognition. *National Institute of Standards and Technology, 803-809.*