DS 4400: Machine Learning and Data Mining 1
Spring 2022
Final Report

Project Title: Capturing the Perfect Smile
Team Members: Andrew Lin, Praneeth Prathi

Code Link: https://github.com/pprathi2018/CapturingSmiles
Video Link: https://youtu.be/o5YsZCTxY9c

Problem Description

The problem that this machine learning project is trying to solve is detecting a smile from a picture. We will be developing supervised classification models that perform binary classification on images to determine whether or not the person in the image is smiling. The practical use of this model can be extended to perform more emotion recognition from pictures, but we will be focusing on smiling. The project's application will be completed with a real-world usage of our classification model by applying the smile detection on a live camera to capture the image when a user is smiling.

References

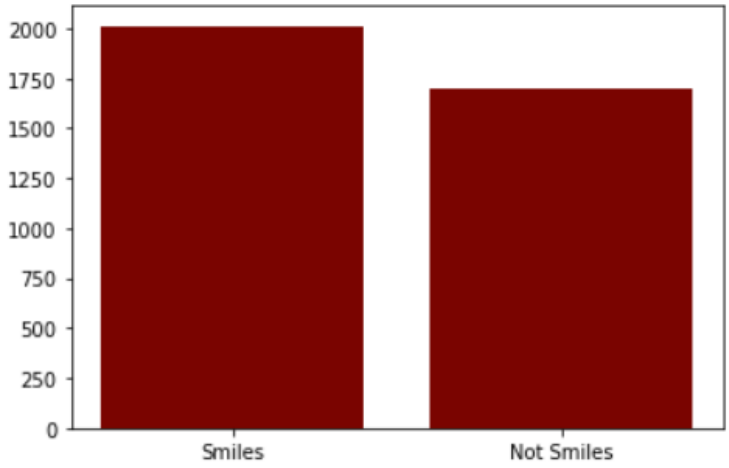
There are many tutorials and resources on packages, models, and algorithms that can be used in facial feature detection. We began by researching the steps of converting facial images to machine-readable contexts, and what models can be best utilized to perform the binary classification. We learned that each image can be converted to a matrix representation, and utilized OpenCV documentation to understand this (https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html). This GeeksForGeeks module describes the exact function we used (https://www.geeksforgeeks.org/python-opencv-cv2-imread-method/). Once we understood how the images are represented, we realized that we may need to crop all of the input images, so that only a person's face is noticed. After doing research, we found out about Haar Cascade models within OpenCV that are used for object and face detection. We learned that these pre-built detection models can be utilized well for our

needs. We studied these articles to understand how Haar Cascades work (https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08#:~:text=So%20what%20is%20Haar%20Cascade,Simple%20Features%E2%80%9D%20published%20in%202001., https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d). There are Haar Cascade algorithms for both facial detection and mouth area detection. This gave us a good starting point to build our models.
For our exploratory data analysis, we wanted to examine the contrast between an image of a smiling face vs a non-smiling face. We referred to this article (https://towardsdatascience.com/exploratory-data-analysis-ideas-for-image-classification-d3fc6bbfb2d2) to understand how to find contrast in pixels between a subset of selected images with a smiling tag of 0 and 1.

Dataset

For our dataset we are using the GENKI-4K dataset of images located here. The MPLab GENKI dataset is a database of images containing faces spanning a wide range of illumination conditions, geographical locations, personal identity, and ethnicity. The dataset contains 4000 human faces classified as non-smiling or smiling by human coders.

For the GENKI-4K dataset, the label distribution is as follows:



The results show that there are 2013 images with examples of a smiling face, and 1697 images without a smile. As our dataset was limited, we did not remove any images with a smiling label, as the distribution is nearly even. We wanted to use all of our available

data in training our models, so we left this distribution and set the necessary weights when building our models.

As the dataset consisted of images, we performed feature extraction and exploratory feature representation by converting the images to their pixel matrix representation. We selected a subset of images from both smiling and non-smiling faces and displayed the average pixel representations.

Average Pixel Distribution of Smiling vs Not Smiling Face

From the average pixel distributions, we can see that the smiling face shows higher obstruction around the mouth with a wider smile and visible cheek bones. This is seen even further in a contrast distribution

Contrast between smiling and not smiling averages

This contrast image shows that the mouth region and corresponding cheek area is the most significant difference between a smiling image and a non smiling face. To analyze this difference even further, we performed PCA extraction from the pixel representation

of the faces to find the specific components that explain 70% of the variability between a smiling face and a non smiling face. The displays of the eigenimages are shown as follows:

Eigenimages after PCA extraction

The eigenimages for the smiling faces show a larger contrast around the mouth and cheek region of the face in comparison to the non smiling images. Based on our exploratory analysis of the selected feature distributions, we understood that the most important features for classifying a smiling vs non smiling image is the pixel distributions around the mouth and cheek area.

Data Preprocessing

The initial goal of our project was to be able to differentiate between a smile and a non-smile in a human face.We read in the original image data using the OpenCV imread() function, to convert the images to pixel value matrix representations.

Without altering any of the original data, we trained a basic Support Vector Classifier model and got very poor results when inputting the data into our model. Based on our exploratory data analysis, we realized that pre-processing the images further would be necessary to extract only the relevant features. We utilized OpenCV's HaarCascade

models which are pre-trained classifiers used to detect various components of a human face, such as the specific face area, or mouth region, etc.

The HaarCascades can be used to crop the images to extract only the face or mouth region. We cropped the face image if it was detected by the HaarCascade and saved the cropped grayscale image to a new directory. If no face was found we saved the original image to the new directory. An example of the Haar Cascade's effect is as follows:

We tried multiple different methods with the cascades to optimize our base model and the results of our trial and error modeling are below. For some trial runs, we used two cascades by additionally running our program on a directory of already cropped photos so that the program might more successfully detect the desired features. For example, in the figure below, "Face + Smile" means we ran the face cascade, saved those files to a directory, and then ran the smile cascade on the cropped face images and used the double cropped pictures as our new data for the model.

Data Alterations / Features	**Training Accuracy**	**Testing Accuracy**
Original	0.5413333333333333	0.538
Face	1.0	0.691
Face + Smile	1.0	0.718
Face + Mouth	1.0	0.6783216783216783

Additionally, we manipulated the parameters of the cascade detectMultiScale() function in detecting cascade features to further improve our data. We changed the values of scaling factors and number of nearest neighbors to find the best numbers for our model. We found many different combinations performed well, but ultimately selected one that performed consistently with reasonable parameters to use.

Our exploratory research indicated that contrasts exist in the entire face area in classifying between a smiling image and non-smiling image. The smile and mouth HaarCascade crops only the mouth region, which may get rid of important features in the cheek area. Therefore, we selected the face HaarCascade to use in pre-processing our base data. We found that in more complex models, the data processed through the face HaarCascade produced better results than just the mouth cascade. We cropped each of our images based on the face HaarCascade, and converted the images to grayscale to get rid of RGB and illumination features. Our dataset with the cropped images can be found here.

Approach and Methodology

We made additional alterations to the dataset on top of using the haar cascades to improve the performance of the models. These are the training and testing accuracies on the base SVC model by removing data points from the dataset that were not croppable by the cascades. The figure below depicts that removing images that weren't croppable by the face cascade resulted in the best testing results for the base SVC model.

Data Alterations / Features	**Training Accuracy**	**Testing Accuracy**
Face + Smile (Base)	1.0	0.718
Remove failed face crop	1.0	0.724
Remove failed smile crop	1.0	0.715
Remove failed crop both	1.0	0.771

For training our models, we split our dataset into a training and testing set using a 75% train-test split. This gives us 3000 images in our training set, and 1000 images in our testing set to validate our models performance. For each of our base classifiers, we utilized GridSearchCV to find the optimal hyperparameters. Since we do not have a significantly large dataset and limited model input, k-fold cross validation is used so that every observation in the data has a chance of being in the training set and testing set. Each fold is held out to validate the model and observe the performance for each of the k folds.

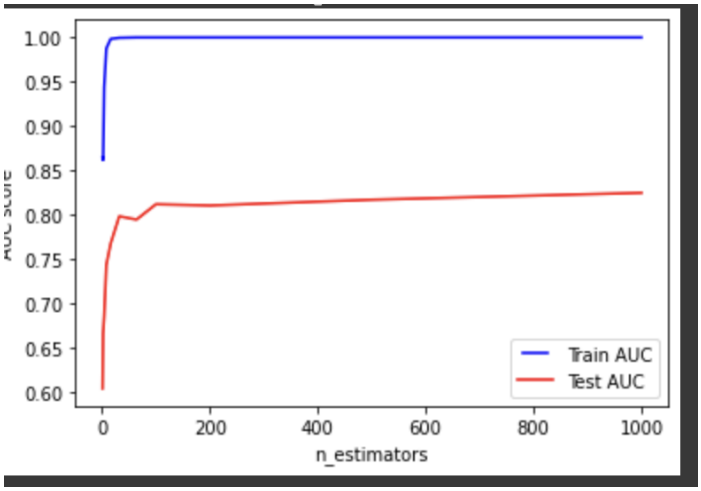
For the SVC, MLP, and Random Forest Classifiers, the input data needs to be flatten pixel representations of the cropped grayscale images. The images are rescaled to be

of size (100, 100), and flattened into an np array, where each input image is represented as an array of pixel values. For the CNN models, the pictures are each represented as a matrix of pixel values. The basic CNN models reads the RGB values of the gray image and converts the shape to be (32, 32, 3). The LeNet CNN model is used for gray-scale images, so the size of the input (32, 32, 1). The cropped images are read using the OpenCV package, and resized to the necessary shape. The AlexNet CNN model is more powerful and utilized for larger images with 3 layers. For this model, OpenCV is used to read the images and convert the shape of the input to be (227, 227, 3). The CNN models do not require flattened data.

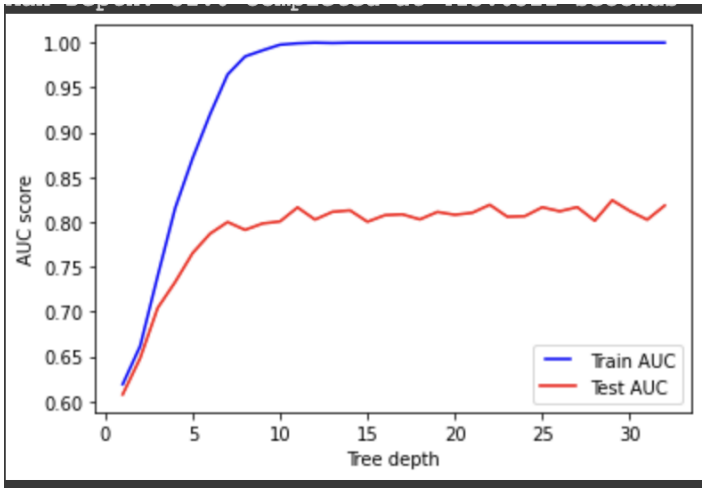
Random Forest Classifier

For our rfc model we followed the same steps to prepare the data that we did for the SVC. We ran gridsearch to identify the best parameters which told us that the optimal parameters for the model were to use 500 n_estimators, auto for max_features, and entropy for the criterion.

Following this, we ran additional tests to check the optimal number of n_estimators and max_depth to reduce any overfitting of our model. After analyzing the results in the figures below, we were able to determine that the optimal n_estimator was between 200 or 500 and selected 200 because it ran significantly faster and lost .001 score on the testing data compared to the 500 n_estimators. For the max_depth, we selected 11 as it had a peak in score and plateaued after. The results of processing our data and optimizing the hyperparameters allowed us to produce a strongly performing model with .820 accuracy.



Performance of RFC model with n_estimators as parameter



Performance of RFC model with max_depth as parameter

Support Vector Classifier

Our Support Vector Classifier was optimized using GridSearchCV to select the optimal hyperparameters for the kernel ('rbf') and C regularization term ('10'). For the regularization term, we tried values of 5 and 10 and found that the higher value lead to better performance.

Multilayer Perceptron Classifier

The MLP classifier is a basic neural network classifier with multiple hidden layers and many neurons stacked together. We utilized GridSearchCV to find the optimal hyper parameters for this scikit-learn model and got an alpha value of 0.001, hidden layer sizes of (20, 20, 20) and the adam optimization algorithm. The model was trained with these hyperparameters on the test set, and k-fold cross validation was performed to analyze the results.

CNN Models

Basic CNN Model

The basic CNN model was our first designed iteration of a CNN model for binary image classification. The model takes in image pixel matrices of size (32, 32, 3) and consists of 3 Convolutional layers, and 2 Max Pooling layers. The output layer of the model uses a sigmoid activation function to predict the results from the weights.

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 15, 15, 32)	0
dropout_6 (Dropout)	(None, 15, 15, 32)	0
conv2d_7 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 6, 6, 64)	0
dropout_7 (Dropout)	(None, 6, 6, 64)	0
conv2d_8 (Conv2D)	(None, 4, 4, 64)	36928
flatten_2 (Flatten)	(None, 1024)	0
dense_4 (Dense)	(None, 64)	65600
dropout_8 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 1)	65

Total params: 121,985
Trainable params: 121,985
Non-trainable params: 0

Basic CNN Model Summary

LeNet CNN Model

For this iteration of the CNN, we mimicked the LeNet model architecture for classification. This model takes in grayscale images of size (32, 32, 1).

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 32, 32, 20)        520

 elu_9 (ELU)                 (None, 32, 32, 20)        0

 batch_normalization_9 (Batc  (None, 32, 32, 20)       80
 hNormalization)

 max_pooling2d_15 (MaxPoolin  (None, 16, 16, 20)       0
 g2D)

 dropout_20 (Dropout)        (None, 16, 16, 20)        0

 conv2d_19 (Conv2D)          (None, 16, 16, 50)        25050

 elu_10 (ELU)                (None, 16, 16, 50)        0

 batch_normalization_10 (Bat  (None, 16, 16, 50)       200
 chNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 8, 8, 50)         0
 g2D)

 dropout_21 (Dropout)        (None, 8, 8, 50)          0

 flatten_7 (Flatten)         (None, 3200)              0

 dense_15 (Dense)            (None, 500)               1600500

 elu_11 (ELU)                (None, 500)               0

 dropout_22 (Dropout)        (None, 500)               0

 dense_16 (Dense)            (None, 1)                 501

=================================================================
Total params: 1,626,851
Trainable params: 1,626,711
Non-trainable params: 140
```

LeNet CNN Model Summary

AlexNet CNN Model

The final CNN model that we developed mimics the AlexNet architecture. This model is more powerful and consists of significantly more layers and trainable parameters. This model takes images in RGB format of size (227, 227, 3).

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 batch_normalization (BatchN  (None, 55, 55, 96)       384
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 27, 27, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       614656

 batch_normalization_1 (Batc  (None, 27, 27, 256)      1024
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 384)       2457984

 batch_normalization_2 (Batc  (None, 13, 13, 384)      1536
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 384)        0
 2D)

 flatten (Flatten)           (None, 13824)             0

 dense (Dense)               (None, 4096)              56627200

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 1)                 4097

=================================================================
Total params: 76,523,137
Trainable params: 76,521,665
Non-trainable params: 1,472
```

AlexNet CNN Model Summary

Discussion and Result Interpretation

Model Performance on Train Set

Model	Accuracy	Precision	Recall	F1	AUC
Random Forest	1.0	1.0	1.0	1.0	1.0
SVC	.995	.997	.993	.995	.995
MLP	0.854	0.864	0.877	0.871	0.851
Basic CNN	0.886	0.891	0.898	0.895	0.885
LeNET	0.981	0.996	0.968	0.982	0.982
AlexNet	0.982	0.973	0.994	0.983	0.981

Model Performance on Test Set

Model	Accuracy	Precision	Recall	F1	AUC
Random Forest	.820	.805	.872	.837	.817
SVC	.818	.832	.843	.834	.815
MLP	.795	.772	.832	.801	.796
Basic CNN	.862	.871	.878	.875	.860
LeNET	.863	.919	.818	.865	.867
AlexNet	.874	.878	.897	.887	.971

The FinalProject.ipynb Jupyter Notebook also displays the confusion matrices for the test and train set along with the generated ROC curve.

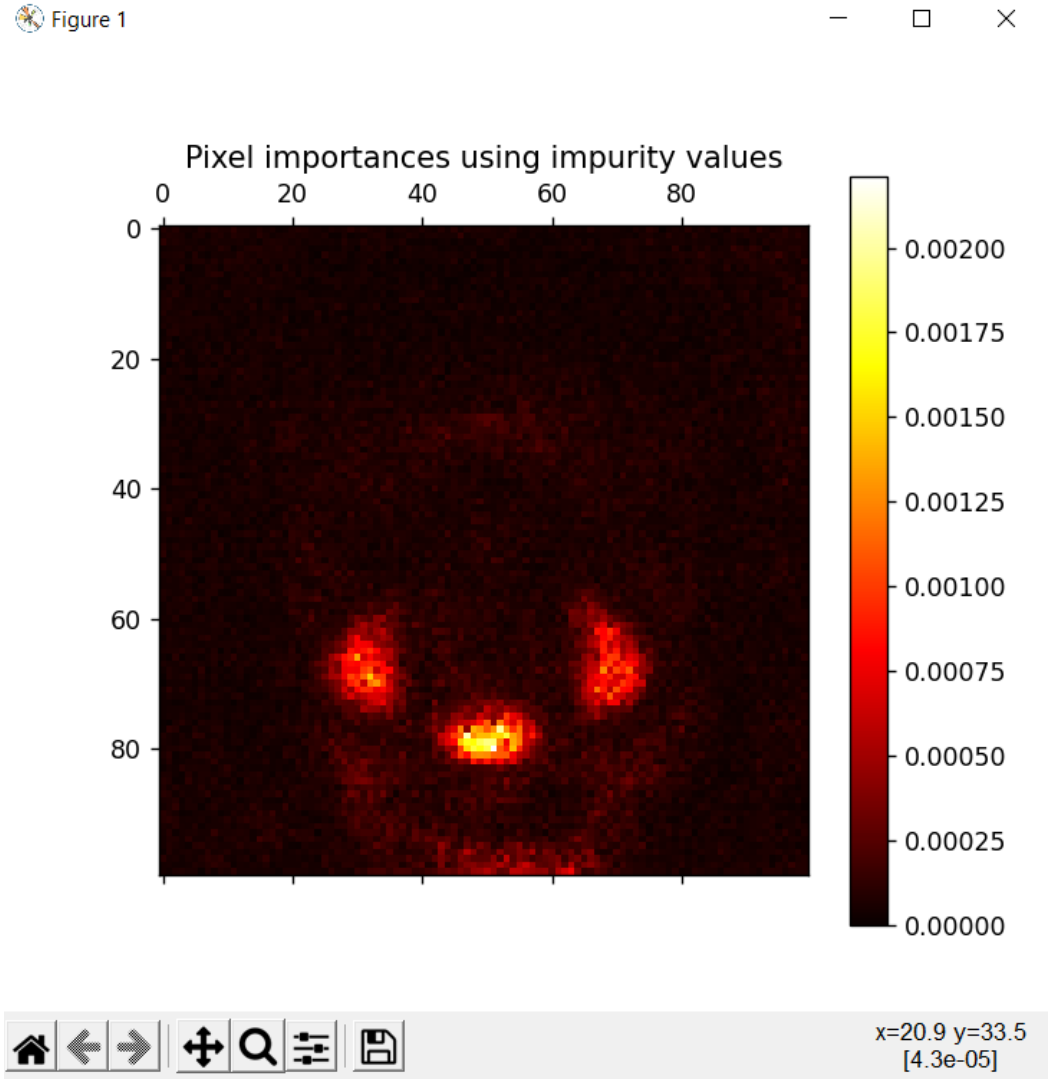
The results show that of the non-CNN models, the Random Forest Classifier performed best on the test set with an accuracy score of 0.820. Random Forest is a Bagging method that builds full grown decision trees on bootstrapped samples of the training data. The number of features used in the decision trees built in the random forest classifier is also randomly selected. This randomness leads to a decrease in the variance of the model. However, the results showed that the RFC model had a 100% accuracy score on the test set. The bagging model makes decisions on a majority vote role and the complexity of the model leads to this 100% accuracy on the train set. Even though we set a max_depth of the trees in the ensemble, the value we chose is still enough to fully build the decision trees and increase our model complexity. Overfitting to the training set may be likely, but the model still performs fairly well on the test set. Similarly, the SVC classifier performed extremely well on the training set with 99.5% accuracy, but performed average on the testing set with an accuracy of 81.8%. This is a result of the model being overfit to the training set. GridSearchCV was used for hyper parameter tuning of the regularization term and the kernel, but these parameters may have also caused the model to be too overfit to the training set. The SVC classifier we used consisted of a rbf kernel, which could have perfectly been able to draw differentiated patterns to separate image features between smiling and non-smiling images in the training set. However, due to the high model complexity, the variance of the model was high, which made the model unable to perform well on the training set. The regularization parameter that we selected should have been higher, so the

complexity of the model decreases. The MLPClassifier is a very basic neural network classifier that performed average for both the training set (85%) and testing set (79.5%). This model did not have significant signs of overfitting, as the model performed similarly on both the test set and the training set.

The CNN models showed a variety of differing results. The basic CNN model performed very similarly on the training set (88% accuracy) and on the test set (86% accuracy). Both the LeNet model and the AlexNet model performed better on both sets, however both models had 98% accuracy on the train set, and only around 87% accuracy on the test set, showing signs of overfitting. This is likely a cause of the complexity of both of these models. In CNNs, overfitting can occur due to the complexity based on the number of convolution and pooling filters in the models, and consequently, the number of trained parameters. The LeNet model has over 1M trainable parameters, while the AlexNet model has over 76M parameters, significantly more than the basic CNN model. Even though the model was overfit, we selected the AlexNet model as our best model due to its performance on unseen data in the test set.

In preparing the dataset for each of our models, we also utilized PCA (Principal Component Analysis) to remove sparsity in our pixel matrix representation of our images. After testing our models with dimensionality reduction from PCA, we found that the models performance did not increase, and in some models, the accuracy decreased. This may be due to the fact that the grayscale images are already cropped to only show the faces of the images. PCA dimensionality reduction is best used to outline necessary and discriminating features in the images, but did not have a significant effect in this case.

Feature Importance



Pixel map feature importance from random forest classifier

The RFC classifier is also used to map feature importances from the images. The pixel importance heatmap shows that the mouth area and cheek area of the face were the most important discriminating features for the random forest model. This result is inline with our exploratory analysis about the discriminating features of the face image.

Conclusion

Using data processing and techniques like GridSearch to optimize hyperparameters and KFold Cross Validation, we were able to optimize all of our models and significantly improve their performance. Overall the neural network models performed better than the others, but the AlexNet model consistently performed the best amongst all of the neural networks and other models with an accuracy of 87.4%. We saved this model using pickle and imported into a video capturing python script that opens up a camera on the screen and takes a snapshot when the user in the camera frame smiles for at least 10 frames. This works by inputting the picture inside of each frame into the neural network to identify if the user is smiling. The program saves frames in grayscale that we reconvert into a rgb picture to recreate the picture of the person smiling.

Future Work

An improvement we can make to our work is to gather additional data to input to our models through a larger dataset. The GENKI-4K dataset consisted of images of people's faces posing for a picture. However, we could use data that is more wild with random snapshots without specific poses. Moreover, a larger dataset could be used to achieve better results for our models. Another improvement that could be made is to test our models on another dataset to ensure the consistency and accuracy of the smile detector.

A lot of our models suffered from overfitting to the training data, so further optimization of hyperparameters is necessary to reduce the complexity of the models, and reduce variance.

Instructions to run VideoCapture

The code to run the smile capture application can be found at the provided link above. The AlexNet CNN model zip file needs to be downloaded from here. This model needs to be extracted into the same directory as the videocapture.py file which then needs to be run to start the video and capture the image. The program will examine each frame to determine a smiling face, and when 10 consecutive frames with a smile are found, a picture will be saved in the results/ folder.

Team Member Contributions

Task	Team Member
Project Proposal	Praneeth, Andrew
Find Dataset	Praneeth
Implement SVC Model	Andrew
Use Haar Cascades to crop images	Andrew
Evaluate model performance	Andrew
Optimize SVC Model 1	Andrew
Optimize cascades	Andrew
KFold Cross Validation	Andrew, Praneeth
GridSearchCV	Andrew, Praneeth
Multi-Layer Perceptron Classifier	Praneeth
Random Forest Classifier Ensemble & Feature Importance	Praneeth, Andrew
CNN, LeNet, AlexNet Models	Praneeth
Compare Model Performances	Praneeth, Andrew
Final Report	Andrew, Praneeth
Video Snapshot App	Praneeth