Facial Expression Recognition on Masked Faces

Yoko Nagafuchi
Stanford University
yokongf@stanford.edu

Zifei Xu
Stanford University
zifei98@stanford.edu

Sally (Hanqing) Yao
Stanford University
yaohanqi@stanford.edu

1 Introduction

Through our project, we tackle the task of facial expression recognition (FER) for images of human faces that are partially covered with a mask. In real life, facial expressions play a key role in communicating with others because they reveal and convey people's emotions and reactions. However, wearing a mask, which has become a norm as we face a global pandemic, has prohibited us from seeing the whole facial expression, such that it has become difficult for us to communicate smoothly. Therefore, in our application-based project, we approach this problem with a FER program that utilizes computer vision techniques to predict expressions, given facial images partially covered with a mask.

The input to our algorithm is a set of color images of human faces, which we pre-process by resizing and applying mask images onto the faces, as well as with other data augmentation techniques. Then, we use 6 types of models to output a predicted emotion: Logistic Regression, Naive Bayes, Random Forest, Support Vector Machine (SVM), K-Means, and Convolutional Neural Network (CNN)-with-ResNet models to classify the images into 7 classes, labeling each face with one of the 7 emotions.

2 Related Work

Karolinska Directed Emotional Faces (KDEF) is a famous dataset for emotional recognition and is commonly used in FER-related studies. In the research conducted by Andrés [**4**], several machine learning techniques were applied to the KDEF dataset to study emotion recognition on full, uncovered faces. He split the dataset into frontal-only and different angles, and for all models, he found that the accuracy for the frontal-only images was higher than that of the other angles. Based on the dataset containing different angles, he achieved an accuracy of 77% for SVM, 55.6% for decision trees, 55% for random forest and 75% for multilayer perceptron.

Especially since 2020, there has been an increasing number of emotional recognition studies on masked faces. One of these studies was conducted using the KDEF dataset. The authors segmented the full images of faces to three partial images: forehead patch, eye patch and skin patch (areas covered by the mask), and trained Inception-V3 network on the different patches separately [**5**]. It was found that the accuracy decreased significantly on the patched images, 44% for forehead patch 52% for eye patch, as compared to 75% on full image. Similar trends were observed on research using other datasets. In a paper by University of Groningen [**6**], the dataset used was masked and unmasked photos taken from German adults. It was concluded that the accuracy dropped from 70% for unmasked photos to 49% for masked photos.

Despite a great number of studies focusing on emotion recognition, the topic is still relatively new with masked faces. In addition, it is challenging to achieve a high accuracy of emotion recognition with masked images due to the fact that masks hide important features on human faces. In our project, we will explore different machine learning techniques to tackle the problem of low accuracy for emotional recognition with masked faces.

3 Dataset and Features

3.1 Dataset

We retrieved a dataset of 4,900 many 762 by 562 pixels images of human faces from the Karolinska Directed Emotional Faces (KDEF) dataset [**1**]. It contains images of 70 people with 7 expressions from 5 angles, all taken in 2 sessions. The images come labeled with one of the 7 emotions: afraid, angry, disgusted, happy, neutral, sad, and surprised, with the code for the particular emotion for an image embedded in its filename. We used this in creating labels for our data.

We applied 7 types of masks to the images in KDEF using Aqeel Anwar's MaskTheFace.py

module [**2**], which detects the facial features on the face in an image and adds a mask image onto the face. The types of masks are surgical, cloth, N95, KN95, gas, black and inpaint. In total, we generated 34,300 images with masks.

Due to constraints in our computational resources, we randomly selected 15,000 images with masks, which photographed 70 people. This dataset contains equal distributions of the 7 emotions and 5 angles. In addition, given that the images had the same background and positioning of the face within each image, we did not extract the faces or facial features to feed into our models.

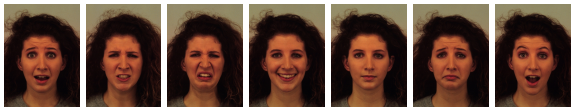

Figure 1: Unmasked 7 different emotions. From left to right: Afraid, Angry, Disgusted, Happy, Neutral, Sad, and Surprised

Figure 2: Different masks applied onto "happy" images, taken from various angles.

3.2 Data Processing

To pre-process the input images, we used the OpenCV library to import each image as a NumPy array, and we resized them to 64 by 64 pixels [**10**]. We normalized each image by dividing each pixel value by 255 so that they are in the [0,1] range. Then, we flattened out the image NumPy array for all models except for the ResNet18 model. By the assumption for the ResNet18 model, the image for this model was further normalized to have mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

We generated a label dataset by mapping the code that is embedded in each image's filename to an integer that represents a specific emotion, and appending the integer to the dataset. Before building our models, we created training and validation datasets for the input data and their labels. We used 5 folds of cross validation such that for each iteration 80% of the data goes to training and 20% is used for validation.

3.3 Data Augmentation

We applied 3 augmentation techniques to our dataset: rotation, flipping, and cropping. We applied functions from OpenCV that rotate the images by 90 degrees clockwise and counter-clockwise, as well as those that flip the images upside down and sideways. For cropping, we cropped out the lower three sevenths of each face, resized them to 64 by 64 pixels, normalized them as with other images, and appended them to our original dataset.

Figure 3: Different transformations applied onto the same happy emotion image. From left to right: 90 degrees clockwise rotation, 90 degrees counterclockwise rotation, 180 degrees clockwise rotation, upside down flipping, left to right flipping, cropped.

4 Methods

We formulated our task to be a multi-class classification problem, training models to predict the label for each image as one of the 7 emotions. Using supervised learning, we attempted SVM, Naive Bayes, random forest, logistic regression, and CNN with ResNet18 models. Briefly, we decided to use these models because they are efficient with processing high dimensional data, and Random Forest is also less prone to overfitting as well as tolerant to outliers [**3**]. Using unsupervised learning, we tried k-means, which converges well on big datasets. The models were implemented using the scikit-learn and PyTorch libraries [**9**][**11**].

4.1 Naive Bayes

We fit a Gaussian Naive Bayes model to predict the label for each training example by assigning the most probable label assuming each class follows a normal distribution. Using Bayes' theorem, this model relies on the assumptions that all pairs of features are conditionally independent given the value of the label, as well as all the features are equally weighted. While the NB assumptions make the model simple and fast to compute, it can also perform poorly since the features of a dataset are usually conditionally dependent.

4.2 Logistic Regression

We fit a multi-class logistic regression model with $L2$ regularization. In this model, we assume that the

conditional probability of the label given a training example follows the softmax function: $p(y = i|x;\theta) = \frac{\exp^{\theta_i^T x}}{\sum_{j=1}^{k} \exp^{\theta_j^T x}}$ - where there are k classes. Regularization helps in reducing the complexity of the model and thus overfitting, by shrinking the coefficients of the model's features. In particular, we added an $L2$ penalty term, the sum of squared magnitudes of the coefficients, to the objective function that we try to minimize to derive θ. We also selected the coefficient λ of the penalty term to be 1, by attempting a range of values on the model using validation dataset. Lastly, we set the maximum number of iterations as 5,000 for convergence.

4.3 Random Forest

A random forest is an ensemble method that consists of a set of decision trees to make predictions. It uses a technique known as bagging, which uses randomly-drawn samples of data, each with size $\sqrt{size\ of\ dataset}$, to train the models on. It also uses the aggregation technique to finalize the prediction, by selecting the class that the majority of the trees predicted. Aggregation leads to an overall stability in the prediction of the label. Additionally, decision trees are built with different subsets of the features, so they are all different. Random forests have two properties that reduce overfitting: bagging and a random subset of features for each split.

We tuned our number of trees within the random forest by using 100, 80, 50, 30, 25, 10, and 5 and observed how the training and validation accuracy fluctuate. Although we would expect a higher number of trees to produce a better prediction, We selected 25 trees in our final model because of its relatively high accuracy score and much shorter runtime compared to the larger numbers of trees.

4.4 K-Means

We used a version of K-means called MiniBatchK-means, which uses mini-batches for training, instead of the entire dataset, that leads to a faster convergence. To choose the optimal values of the number of clusters *k* and batch size *b*, we chose the highest accuracy score of models on the validation set for a range of values. We attempted values of *k* from 7 to 25 because we have 7 distinct true labels. We attempted *b* ranging from 6 to 2400. For $k > 7$, we used a function that maps the predicted labels in [0, k] to labels within [0, 6].

Similar to plain K-means, the aim is to minimize the distortion function:

$$J(c, \mu) = \sum_{i=1}^{n} ||x^{(i)} - \mu_{c^{(i)}}||^2,$$

which measures the sum of squared distances between each training example $x^{(i)}$ and the centroid it is assigned to: $\mu_{c^{(i)}}$. Minimizing *J* means assigning all training examples to their closest centroids.

4.5 SVM

We applied a multinomial SVM with a nonlinear kernel. The goal of SVM is to find a hyperplane with the maximum margin that separates the dataset into multiple classes. In particular, we used the one-versus-one approach, which trains an SVM for each pair of classes and assigns a predicted label to each training example by selecting the class that the point was assigned to the most by the generated SVM's.

To choose the optimal values of the regularization and kernel coefficients, *C* and γ, as well as the kernel type, we attempted the following values and selected the ones that produced the highest accuracy score on the validation set: $C = [0.1, 1, 10, 100], \gamma = [0.0001, 0.001, 0.1, 1]$, kernel: Radial Basis Function and 3rd-degree Polynomial. Below is the formula for Radial Basis Function:

$$K(x, x') = exp(-\frac{||x-x'||^2}{2\sigma^2}),$$

4.6 CNN with ResNet18

We applied a ResNet18 network to our convolutional neural model, a technique known as transfer learning. ResNet18 has been pre-trained on the ImageNet dataset containing over 15 millions of images in 1000 categories [**8**].

We had originally trained a CNN model from scratch, which turned out to perform poorly, about the same as our baseline Naive Bayes model [**12**]. Transferring some features from previously-learnt models that were used to perform similar tasks can speed up the process of training the CNN model as well as improve the performances of models, especially when the current dataset is small [**7**]. We used a batch size of 60 and epoch size of 30. We also added a fully connected layer with 7 classes for the final output of our classification task. We transformed the data using torch library before feeding it into our NN.

5 Results and Discussion

5.1 Baseline Model

We used Naive Bayes as our baseline model because it has faster convergence compared with other models and thus could help us quickly evaluate the way

we preprocessed the images. Fitting the model on a random 80% and 20% split of approximately 15,000 images without cross validation gives us 25.84% training accuracy and 24.48% validation accuracy. The relatively low accuracy indicates that the model has high bias, is too simple, and its Naive Bayes assumptions that features are conditionally independent and each class is normally distributed are likely not the case for our data. An interesting finding from this model is that from the confusion matrix in Figure 4, happy has the highest accuracy and highest false positive rate among all emotions.

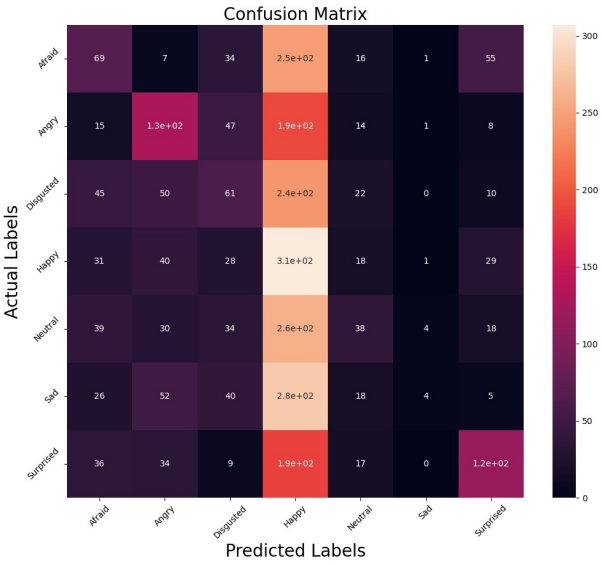

Figure 4: Confusion Matrix from Naive Bayes

5.2 Metrics

We evaluate the models based on the average accuracy and mean squared error obtained from 5-fold cross validation. Only the ResNet18 model is evaluated by accuracy and loss. We chose to use accuracy, which is the ratio of the correct predictions to all predictions, because we most cared about whether our models predicted the emotion correctly or not, our dataset was balanced in terms of the distribution of the classes, and the labels were equally important.

From Figure 5 we see that K-Means has the lowest accuracy. This is expected as K-Means does not work well on high dimensional data because it minimizes the Euclidean distances between points and centroids. This issue could potentially be alleviated by applying Principal Component Analysis to the images for dimension reduction before fitting it to K-Means. Naive Bayes also did not work well, as mentioned in section 5.1, due to the strong assumptions of NB. The best performing models are Random Forest and Logistic Regression. This is because Random Forest uses an ensemble of decision trees, which outperforms individual classifiers most of the time. It was also observed that random forest takes a shorter time than logistic regression to train and make predictions, due to parallel processing of the trees. For logistic regression, it makes very weak assumptions about the dataset and generally performs well on classification tasks. For SVM, the accuracy is high but the efficiency was outstandingly low. Due to limited computing resources, we were unable to train SVM on 15k images as other models. Instead, we trained one tenths (around 1.5k) of images with cross validation for SVM, and it took around 24 hours to finish training and prediction, which is more than 100 times longer compared to a few minutes for random forest.

Model Type	Accuracy		MSE	
	Training	Validation	Training	Validation
Random Forest	100.00%	99.99%	0.0000	0.0024
Logistic Regression	100.00%	99.59%	0.0000	0.0594
Naïve Bayes	24.47%	23.86%	5.3018	5.3591
K-Means	17.82%	19.49%	7.8376	7.6617
SVM	100.00%	99.84%	0.0000	0.0321

Figure 5: Metric Results[1]

The accuracy and loss for ResNet18 against 30 epochs is shown in Figure 6. Accuracy strictly increases over the epochs with training accuracy reaching 100% and validation accuracy reaching 70%. This indicates ResNet18 has the tendency to overfit, but a better validation accuracy can be achieved by increasing the epochs, which trades off time and computing resources.

5.3 Effects of Data Augmentation

We randomly selected about 2,950 images to apply each of the rotation, flipping, and cropping transformations. We performed cross validation to evaluate the effects of augmenting the data on the Random Forest and Logistic Regression models. Average accuracy and average mean squared errors are shown in Figure 7. As we can see from the results, augmenting the images does not improve the accuracy and produces higher MSE on both models, which could indicate that emotion recognition, especially masked and object oriented emotion recognition, is more image orientation dependent.

[1]SVM was trained and tested on one-tenths of the number of images that all other models were trained on (15k).

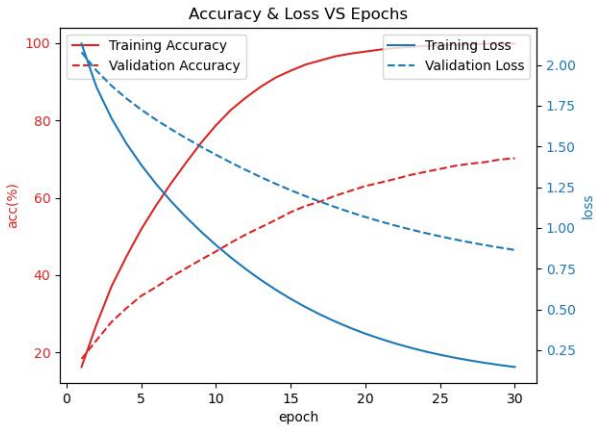

Figure 6: Accuracy and Loss VS Epochs for ResNet18.

Model Type	Data Type	Validation Accuracy	Validation MSE
Random Forest	Unaugmented	100.00%	0.000
Random Forest	Augmented	99.95%	0.000
Logistic Regression	Unaugmented	99.59%	0.062
Logistic Regression	Augmented	84.89%	1.428

Figure 7: Effects of Data Augmentation

6 Conclusion/Future Work

Based on our 6 models, we found that random forest and logistic regression without data augmentation were the highest performing in terms of accuracy and computational efficiency. We also found that Naive Bayes and K-Means did not perform well with respect to the same measures because of their assumptions about the data and simplistic models. CNN with ResNet18 performed better because of transfer-learning, using the pre-trained parameters to overcome the small dataset. We cannot directly compare SVM with the other models because we could only train the model on 10 folders of images, due to the limit of our computational resources. If we were to compare SVM directly to the other models, SVM was the second best in its accuracy. However, it was the worst in its computational efficiency - it took around 24 hours to complete the program, even with a smaller dataset.

One limitation of the project is that the performance of our models is low on faces that we have never seen. Our validation data is generated by a 20% split from the whole dataset, which contains a lot of images of the same people. Hence, the faces in our validation dataset is likely already have been trained when fitting the model, either with a different mask or different emotion. When we constructed a test dataset with completely new faces and feed them to the model, we found that the accuracy was a lot lower than the validation accuracy. Moreover, the KDEF datasets only contains images of Swedish adults. It will have limited performance when predicting for other populations, for example, Asians or children. Future works to be done to tackle this issue could be including more images with diverse demographic features. By doing this, we expect to see less overfitting and better performance on unseen faces. However, this would also increase the dataset significantly and more computing resources would be needed.

Another technique that we have not tried on the data is feature selection, for example, partitioned-feature-based classifier or Principle Component Analysis. These would reduce dimensionality and alleviate overfitting, as well as reduce the computing time for a model like SVM. In addition, future researchers could also try other pre-trained CNN models, such as VGG19, EfficientNet and InceptionV3. Lastly, other versions of ResNet with more layers could also be used, as well as adding our custom layers before the output layer to those pre-trained models.

7 Contributions

The code for our project can be found here: `https://github.com/sallyyaohanqing/CS229_Project.git`.

The three of us worked together on formulating the problem, finding the dataset, and writing the report. Each of us mainly focused on:

HY worked on much of the coding for creating training and test datasets and building the different models using ML libraries. Trained Random forest, CNN. Tabulated results and created plot for confusion matrix. Set up github repository for the project.

ZX worked on applying the mask library to all of our images, setting up Google Colab. Trained Resnet18 and Logistic Regression. Created plot for Resnet18.

YN worked on K-means, Naive Bayes, and SVM, as well as the writing for the final report.

References

[1] Lundqvist, D., & Litton, J. E. (1998). The Averaged Karolinska Directed Emotional Faces - AKDEF, CD ROM from Department of Clinical Neuroscience, Psychology section, Karolinska Institutet, ISBN 91-630-7164-9.

[2] Aqeel Anwar: MaskTheFace - Convert face dataset to masked dataset https://github.com/aqeelanwar/MaskTheFace

[3] Horning, Ned. Random Forests: An Algorithm for Image Classification and Generation of Continuous Fields Data Sets. American Museum of Natural History Center for Biodiversity and Conservation, http://wgrass.media.osaka-cu.ac.jp/gisideas10/papers/04aa1f4a8beb619e7fe711c29b7b.pdf.

[4] Andrés, J. Machine Learning and Deep Learning for Emotion Recognition. https://upcommons.upc.edu/bitstream/handle/2117/184076/tfm-joan-sisqella.pdf?sequence=1&isAllowed=y.

[5] Saravanan, P., & Ravindran, S. (2021). A Performance Study on Emotion Models Detection Accuracy in a Pandemic Environment. https://link.springer.com/chapter/10.1007/978-3-030-90235-3_28.

[6] Grundmann, F., & Epstude, K., & Scheibe, S. (2021). Face masks reduce emotion-recognition accuracy and perceived closeness. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8064590/

[7] Singhal, G. (2020). Transfer Learning with ResNet in PyTorch. https://www.pluralsight.com/guides/introduction-to-resnet

[8] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[9] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, 2011, pp. 2825-2830.

[10] The OpenCV Library, Bradski, G., Dr. Dobb's Journal of Software Tools, 2000.

[11] PyTorch: An Imperative Style, High-Performance Deep Learning Library, Paszke, Adam, et al. *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[12] Sanad. (2020). Learn Image Classification on 3 Datasets using Convolutional Neural Networks (CNN). https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/