Learning to Recognize Facial Expressions

Tao Li
Department of Computer Science
li2953@purdue.edu

Tinghan Yang
Department of Computer Science
yang1683@purdue.edu

Zihan Wang
School of Mechanical Engineering
wang5044@purdue.edu

Abstract

In this work, we comprehensively review automatic methods for facial expression recognition. We first explore four canonical machine learning algorithms, including Perceptron, Adaptive Boosting (AdaBoost), Decision Tree, and Support Vector Machine (SVM), and perform hyperparameter tuning under different experimental settings and image encoding methods. We then evaluate their performances on a benchmark dataset by leveraging two resampling techniques for validation (i.e., cross-validation and boostrapping). To gain deeper insight of the models, we apply various visualization techniques, including ROC curve and confusion matrix, and investigate relationships between the hyperparameters and models verified by rigorous statistical tests. We further compare the fine-tuned models with the state-of-the-art neural network architectures (e.g., VGG and ResNet) and suggest directions for future improvements.

1 Introduction

Facial expressions play a key role in human communication and social interaction. Due to its importance, automatic facial expression recognition attracts increasing attention over years [Shan et al., 2009, Pantic and Rothkrantz, 2000, Fasel and Luettin, 2003, Essa and Pentland, 1997]. In 1970s, Ekman and Friesen [1971] defined six basic emotions based on a cross-cultural study [Ekman, 1994], which suggests that humans perceice the same basic emotions (i.e., anger, disgust, fear, happiness, sadness, surprise) regardless of cultural backgrounds. Fig. 1 shows examples of these basic emotions from the dataset we use in this project.

Traditionally, facial expression recognition (FER) systems use handcrafted features from still facial images (e.g., local binary patterns (LBP) [Shan et al., 2009], LBP on three orthogonal planes (LBP-TOP) [Zhao and Pietikainen, 2007], non-negative matrix factorization (NMF) [Zhi et al., 2010], and sparse learning [Zhong et al., 2012]). Recently, with the advances in computing power (e.g., popularity of GPUs) and well-designed neural network architectures, machine learning algorithms especially deep learning-based methods have achieved the state-of-the-art and outperformed traditonal methods by a large margin [Krizhevsky et al., 2017, Simonyan and Zisserman, 2014, Szegedy et al., 2015, He et al., 2016]. Although has been studied extensively, automatic expression recognition in the wild and with a high accuracy is still a challenging task given by complexity of human faces and subtlety and variability of facial expressions. We refer Li and Deng [2020] for a more comprehensive review.

Figure 1: Examples of images in the FER2013 dataset Goodfellow et al. [2013]. Each column illustrates one of the seven facial expressions: anger, disgust, fear, happiness, sadness, surprise, and neutral. Even under the same category, the images may vary in pose, illumination, orientation, background, and so on, which make the dataset challenging.

Our contributions are multi-fold:

- We explored four classic machine learning algorithms (i.e., Perceptron, Decision Tree, AdaBoost, and SVM), each of which we examed at least three sets of hyperparameters;
- We compared them with three state-of-the-art deep learning models re-trained on the same benchmark dataset;
- For fair comparison, we leveraged two resampling techniques for validation (i.e., cross-validation and boostrapping);
- We adopted two image encoding methods, including raw pixels and geometric features Incorporated with facial landmarks, and show that adopting facial landmarks enhances model performance verified by statistical significance test;
- We applied various visualization techniques, such as ROC curves and confusion matrices, to better understand the models and the dataset;
- We investigated the relationship between accuracy and the size of training samples as well as iteration times.

The rest of the report is organized as follows: in Section 2 we briefly review the dataset in use; we then review serveral machine learning algorithms (as specified in the project plan) in Section 3; Section 4 details experimental settings and hyperparameters, and demonstrates the experimental results; in Section 5, we further compare the results under different settings and analyze both qualitatively and quantitatively; this report is concluded in Section 6 with discussion of future works.

2 Dataset

FER-2013 dataset 2013 was created by Pierre Luc Carrier and Aaron Courville, by using Google image search API to search for images of faces that match a set of 184 emotion-related keywords like "blissful", "enraged", etc. Together with keywords like gender, age and ethnicity, about 600 strings were used to query facial images, and the first 1000 images return for each query were kept for the next stage of processing.

The collected images were approved by human labelers who removed incorrectly labeled images, cropped to only faces by bounding box utility of OpenCV face recognition, and resized to 48 x 48 pixels greyscale images. Then a subset of the images were chosen by Mehdi Mirza and Ian Goodfellow, and the labels(categories) of the chosen images were also mapped from the fine-grained emotion keywords.

The resulting FER-2013 dataset contains 35887 images with 7 categories in total. Specifically, there are 4953 "Anger" images, 547 "Disgust" images, 5121 "Fear" images, 8989 "Happiness" images, 6077 "Sadness" images, 4002 "Surprise" images, and 6198 "Neutral" images, with label ids ranging from 0 to 6 (see Fig. 2).

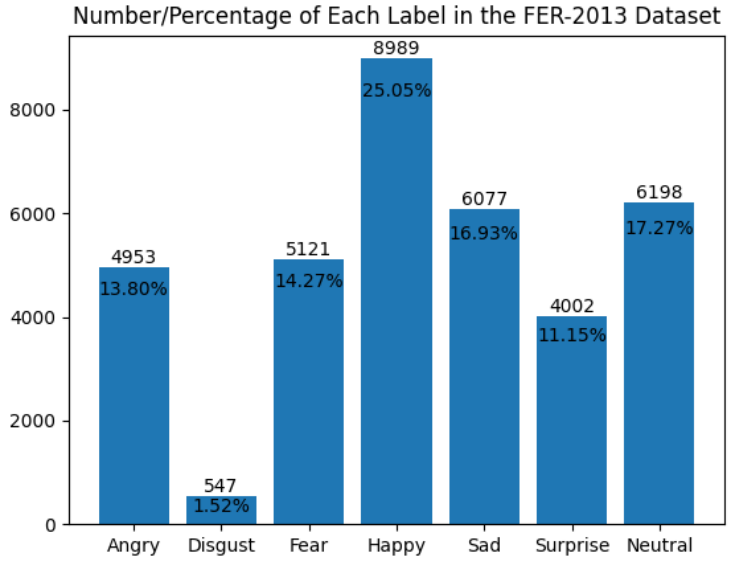


Figure 2: Distribution of labels in FER-2013 Goodfellow et al. [2013] dataset. The seven labels (i.e., anger, disgust, fear, happiness, sadness, surprise, and neutral) not evenly distributed among the 35887 samples: *happiness* is the most popular expression with almost 9000 samples; in comparison, *disgust* has only around 500 samples.

Goodfellow et al. [2013] have shown that the potential label errors in FER-2013 dataset do not make the classification problem significantly harder due to the experimental result that human accuracy on a small-scale dataset with 1500 images, 7 expression categories and no label error is 63-72%, which is very close to the human accuracy for FER-2013, which is 60-70%.

3 Method

In this section, we outline three classes of algorithms used in the project, i.e., Perceptron Algorithm, Adaptive Boosting (AdaBoost), and Support Vector Machine (SVM), including their basic models and variants (e.g., combinations of different regularization terms, constants). Details of hyperparameter tuning and experimental settings will be in Section 4.

3.1 Perceptron Algorithm

Perceptron algorithm [Rosenblatt, 1957] is a linear model that learns a binary classifier in a supervised manner. Formally, given input vector $\mathbf{x}$, a single-class perceptron outputs $f(\mathbf{x})$:

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & otherwise \end{cases} \tag{1}$$

where $\mathbf{w}$ is weight vector and b is bias. It can be naturally generalized to a multi-class classifier. For input $\mathbf{x}$ and label y of an arbitrary sample from training set, we use a feature function $f(\mathbf{x}, y)$ that maps the pair to a feature vector. Then the problem is converted to

$$\hat{y} = \underset{y}{\operatorname{argmax}} f(\mathbf{x}, y) \cdot \mathbf{w}. \tag{2}$$

And the update formula for sample t becomes:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + f(\mathbf{x}, y) - f(\mathbf{x}, \hat{y}). \tag{3}$$

Note that there are plenty of variants of perceptrons derived from this standard form; for example, the choice of regularization term, the constant that multiplies the regularization term, and number of iterations. We will compare these hyperparameters quantitatively in Section 4. Here, we provide their formal definitions for reference.

Regularization terms. Given feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, popular choices of the regularization term (i.e., penalty) of Perceptron algorithm are L1-norm, L2-norm, and elastic net [Weisstein, 2002, Zou and Hastie, 2005]. L1-norm:

$$||\mathbf{x}||_1 := |x_1| + \cdots + |x_n|. \tag{4}$$

L2-norm (Euclidean norm):

$$||\mathbf{x}||_2 := \sqrt{x_1^2 + \cdots + x_n^2}. \tag{5}$$

Elastic Net:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} ||y - \mathbf{x}\beta||_2^2 + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1. \tag{6}$$

We refer Zou and Hastie [2005] for more details of Elastic Net regularization.

3.2 Adaptive Boosting

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases, introduced in 1995 by Freund and Schapire [1997].

In principle, AdaBoost algorithm is trying to estimate the ensemble

$$h_m(\underline{x}) = \alpha_1 h(\underline{x}; \underline{\theta_1}) + \cdots + \alpha_m h(\underline{x}; \underline{\theta_m}) \tag{7}$$

by minimizing the training loss

$$\sum_{t=1}^{n} Loss(y_t h_m(\underline{x_t})) \tag{8}$$

with respect to the non-negative votes (α_i) This is a hard problem to solve jointly, but base learners ($\mathbf{h}(\underline{\mathbf{x}}; \underline{\theta}_\mathbf{m})$) can be added sequentially (forward fitting) to minimize the loss.

The AdaBoost alrorithm used in this project is SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function), which is a multi-class extension of two-class AdaBoost algorithm, and does not need to be reduced to multiple two-class problems . SAMME (and its variant SAMME.R) shares the same modular boosting structure of AdaBoost, i.e. adaptively combining weak classifiers into a powerful one, but only requires the performance of each weak classifier be better than random guessing 1/K (K is number of labels) instead of 1/2. SAMME was developed by Hastie et al. [2009], and is proved to have comparable or even better performance with that of classic AdaBoost algorithms.

The main parameters to tune to obtain good results are max number of weak classifiers and the learning rate. The former parameter decides the maximum base estimators at which the boosting is terminated, and the latter one shrinks the contributions of each classifier. In the experiments we fixed the base learner to be single-layer Decision Tree Classifier (decision stump introduced in the class) to focus more on the boosting performance itself regardless of base estimator types [Pedregosa et al., 2011].

3.3 Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

From literatures [Safavian and Landgrebe, 1991, Friedman et al., 2001], we know that some hyperparameters of decision tree are crucial to its performance. For example,the maximum depth of the tree and the minimum number of samples to split a node, as well as split strategy and the criterion to evaluate a split, where usually Gini Index [Lerman and Yitzhaki, 1984] and Information Entropy [Shannon, 1948] are used. We will explore these different settings in Section 4 to gain more insights.

3.4 Support Vector Machine

Support Vector Machine (SVM) is kind of supervised learning algorithm for classification and regression analysis. It represents the data samples as points mapped into a kernel space, and then divides those data points by a hyperplane which aims at maximizing the gap between the plane and the data points of different categories. When a new data sample comes, the SVM model will predict its class based on the side of the hyperplane it falls into.

To be specific, the mathematical problem that we are going to solve for SVM is a optimization problem (duo SVM)

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \cdots n \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

The hyperparameters that we are going to tune include:

- C, which is the coefficient of slack variables and represents the scale of the penalty term;
- K, which is the type of the kernel we use, such as linear kernel, polynomial, radius basis function, sigmoid, etc.
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$.
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + b)^n$
  - Radial basis function kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$, in which we will also tune the value of γ.
  - Sigmoid kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(k\mathbf{x}_i \cdot \mathbf{x}_j + b\right)$

3.5 Deep Neural Network

With the popularity of high performance computing devices (e.g., GPUs) and advances in network architecture designs, deep learning-based algorithms have be widely used and shown impressive performances in the domain of computer vision, epsicially in the task of image classification. In this project, we explore three canonical deep neural network architectures: simple Convolutional Neural Network [LeCun et al., 1995] (in short, CNN or ConvNet), “very deep” convolutional neural networks [Simonyan and Zisserman, 2014] (VGG), and Residual Neural Network [He et al., 2016] (ResNet). Especially, VGG and ResNet are the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014 and 2015 respectively. Fig. 3 demonstrates examples of neural network architectures. For this project, we adopt the three network architectures (with minor adjustment) and retrain them on our own dataset.

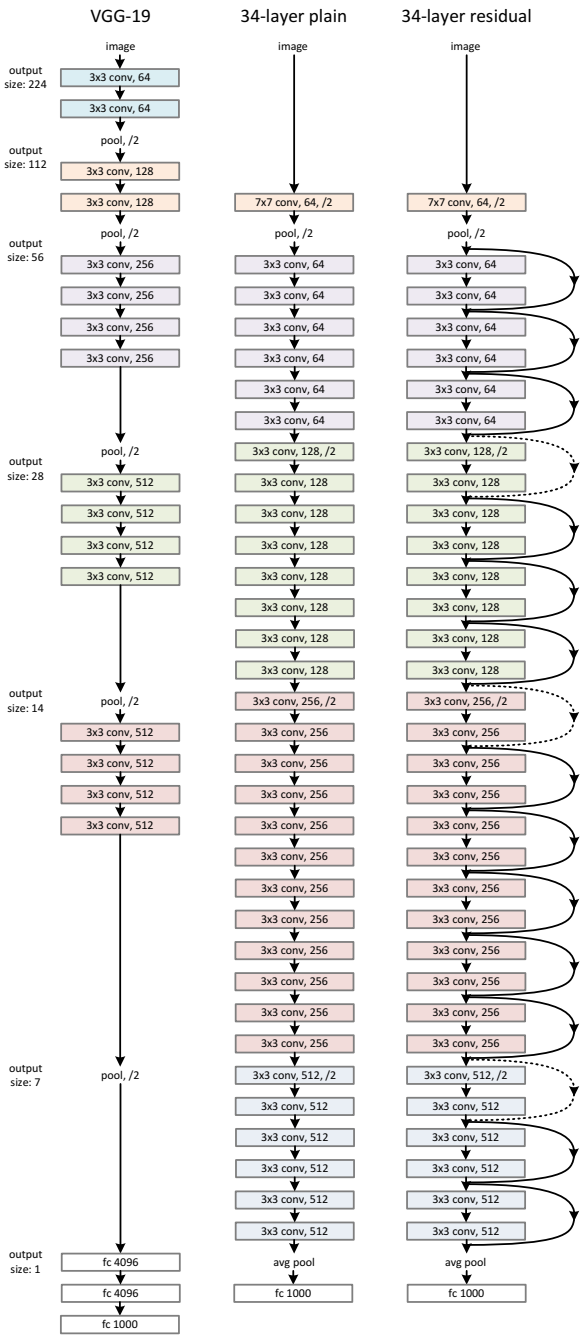


Figure 3: Examples of Neural Network Architectures [He et al., 2016]. Left is a VGG model [Simonyan and Zisserman, 2014] with 19 layers; Middle is a plain neural network with 34 layers; right is a Residual Network (with residual blocks [He et al., 2016]).

4 Experiment

In this section, we details our experimental settings and presents the results. First, we introduce the dataset setup in Section 4.1. Then, we present two image encoding methods we used in Section 4.2. Selection of features and hyperparameters are critical to the performance of an algorithm. In Section 4.3, we provides the details of hyperparameter tuning of four canonical machine learning algorithms (i.e., Perceptron, Decision Tree, Adaptive Boosting, and Support Vector Machine). To fairly evaluate the algorithms, we conduct experiments under different settings and use resampling

techniques for validation (i.e., cross validation and boostrapping) which is described in Section 4.4. We present numerical results of the aforementioned algorithms in Section 4.5 along with rudimentary evaluation. We leave more in-depth discussions in Section 5.

4.1 Dataset Setup

As mentioned in Section 2, the original dataset has more than 35000 samples with imbalanced distributed labels. For simplicity, we pick 500 samples for each expression category (see Fig. 1), which forms a subset with a total of $500 \times 7 = 3500$ samples.

4.2 Image Encoding

In previous discussions, we assumed that the feature vector of an image is given. In practice, image coding (i.e., mapping of a 2D grayscale image to a feature vector) plays an crucial role in the performance of the algorithms. So far, we have explored two image encoding methods: raw pixels and facial landmarks.

Raw Pixels. The original images in FER2013 is in 48×48 resolution and grayscal. Each of the pixel is an integer ranging from 0 to 255. For this simpliest encoding, we flatten the 2D image matrix to 1D array and normalize it to $[0, 1]$. Then we obtain the feature vector of length 2304.

Facial Landmarks. By leveraging geometric features of faces and a pre-trained deep model, we obtain 68 facial landmarks for each image (see Fig. 4 for examples), which provide us with extra semantic information of the faces from the raw pixels representation. Specifically, we can extract certain geometric features of faces from these landmarks which, intuitively, may relate to facial expressions (e.g., the size of mouth, distance between two lips). Here, we follow Khan [2018] and convert the 68 landmarks to 12 features. Table 1 shows details of the conversion, where p_i represents the i-th landmarks and $||p_i - p_j||$ denotes the Euclidean distance between points i and j.

Figure 4: Examples of facial landmarks. The first row shows the original images in FER2013 [Goodfellow et al., 2013]; the second row are added with 68 facial landmarks.

4.3 Hyperparameter Tuning

In this project, we finely tune four classic machine learning models and then compare them with three state-of-the-art deep learning-based methods. During the tuning process, we borrow the idea of k-fold cross validation and get the mean accuracy on all k folds. The details of k-fold cross validation will also be discussed later.

Followings are the details of our hyperparameter tuning.

Perceptron Algorithm. For perceptron algorithm, we consider three sets of hyperparameters: regularization terms, number of iterations, and the constant α that multiplies the regularization

Table 1: Facial landmarks to feature vector.

Feature	Description
$\|\|p_{37} - p_{40}\|\|$	Left eye height
$\|\|p_{38} - p_{42}\|\|$	Left eye width
$\|\|p_{43} - p_{46}\|\|$	Right eye height
$\|\|p_{44} - p_{48}\|\|$	Right eye width
$\|\|p_{18} - p_{22}\|\|$	Left eyebrow width
$\|\|p_{23} - p_{27}\|\|$	Right eyebrow width
$\|\|p_{49} - p_{55}\|\|$	Lip width
$\|\|p_{18} - p_{41}\|\|$	Distance between left eye and left eyebrow
$\|\|p_{48} - p_{27}\|\|$	Distance between right eye and right eyebrow
$\|\|p_{34} - p_{67}\|\|$	Distance between nose centre and lips centre
$\|\|p_{42} - p_{49}\|\|$	Distance between left eye and lips left corner
$\|\|p_{47} - p_{55}\|\|$	Distance between right eye and lips right corner

Table 2: Tuning the Hyperparameters of Perceptron Algorithm

Penalty	L1			L2			Elastic Net		
Iteration	100	500	1000	100	500	1000	100	500	1000
α = 0.01	0.216	0.231	0.241	0.150	0.148	0.144	0.152	0.149	0.145
α = 0.001	0.253	0.233	0.238	0.177	0.163	0.191	0.160	0.190	0.182
α = 0.0001	0.269	0.239	**0.272**	0.221	0.212	0.201	0.206	0.213	0.226

terms. As discussed in Section 3.1, L1, L2, and Elastic Net are the three most popular selections for regularization. In Table 2 we show the accuracy under these three settings and find that using L1-norm for regularization with $\alpha = 0.0001$ provides the best result. We hereby use this setting for the following task.

Table 3: Hyperparameter Tuning of SVM

Kernel	Linear		Polynomial		Radial Basis		Sigmoid	
Shape	OVO	OVR	OVO	OVR	OVO	OVR	OVO	OVR
C = 0.01	0.431	0.431	0.254	0.254	0.269	0.269	0.134	0.134
C = 0.1	0.377	0.377	0.286	0.286	0.254	0.254	0.134	0.134
C = 1	0.314	0.314	0.389	0.289	0.343	0.343	0.151	0.154
C = 10	0.311	0.311	0.430	0.430	**0.434**	**0.434**	0.183	0.154
C = 100	0.314	0.314	0.377	0.377	0.380	0.380	0.146	0.106

SVM. When tuning the hyperparameters of SVM, we consider the combinations of the following parameters, kernel function, the shape of decision function, and the parameter of penalty term. In our experiment, we explore 4 kernel functions (including `linear, poly, rbf, sigmoid`), 7 penalty parameters (including `C = 0.01, 0.1, 1, 10, 100`), and 2 decision function shapes (`decision_function_shape = 'ovo', 'ovr'`, where `ovo` means "one-versus-one", i.e., for n classes there are $n(n-1)/2$ two-class classifiers, and `ovr` means "one-versus-rest", i.e., there are n classifiers, each of which regards one class as positive class and the rest classes as negative classes).

Table 4 gives the mean accuracy of SVM on a randomly splitted training set and test set, it can be seen that `rbf` kernel with `C = 10, kernel='rbf'` achieves the best accuracy. In order to better tune the `rbf` kernel, we further explore the value combinations of C and γ. The results are shown in Table 4. Note that the default value of γ is `'scale'`, which can be calculated as $\gamma = 1/\left(n_{features} * \text{Var}(X)\right)$, while $\gamma =$ `'auto'` means $\gamma = 1/n_{features}$. In the following experiments, we just apply the best hyperparameters combination, i.e., `C = 10`, γ `= 'auto'`, `kernel = 'rbf'`, which is corresponding to the highest accuracy 0.437 in Table 4.

Table 4: Hyperparameter Tuning of SVM (kernel = `rbf`)

	$\gamma = 0.01$	$\gamma = 0.1$	$\gamma = 1$	$\gamma = 10$	$\gamma = 100$	$\gamma =$'auto'	$\gamma =$'scale'
C = 0.01	0.269	0.163	0.154	0.154	0.149	0.271	0.269
C = 0.1	0.294	0.163	0.154	0.154	0.154	0.271	0.254
C = 1	0.391	0.211	0.177	0.177	0.177	0.357	0.343
C = 10	0.351	0.22	0.177	0.177	0.177	**0.437**	0.434
C = 100	0.351	0.22	0.177	0.177	0.177	0.374	0.38

Decision Tree. For decision tree, as discussed in Section 3.3, we select three important hyperparameters: criterion, minimum number of samples that can split a node, and the maximum depth of a tree. Table 5 shows a slight trend that the accuracy increases with the maximum depth of a tree, and when we are picking Information Entropy for criterion, *min_sample_split* as 4, and *max_depth* as 16, we obtain the best set of hyperparamters under this setting.

Table 5: Hyperparameter Tuning of Decision Tree

Criterion	Gini					Entropy				
Min Sample Split	2	4	8	16	32	2	4	8	16	32
Max Depth = 2	0.243	0.243	0.243	0.243	0.243	0.243	0.243	0.243	0.243	0.243
Max Depth = 4	0.275	0.275	0.275	0.275	0.275	0.268	0.268	0.268	0.268	0.268
Max Depth = 8	0.260	0.261	0.257	0.256	0.259	0.263	0.265	0.264	0.262	0.258
Max Depth = 16	0.264	0.262	0.266	0.259	0.260	0.264	**0.276**	0.264	0.260	0.262
Max Depth = 32	0.263	0.264	0.264	0.260	0.263	0.269	0.271	0.268	0.256	0.262
Max Depth = ∞	0.266	0.267	0.260	0.264	0.258	0.267	0.266	0.272	0.250	0.260

AdaBoost. When tuning hyperparameters of AdaBoost, we choose 5 values for Learning Rate (0.001, 0.01, 0.1, 1.0, 10.0), and 10 values for maximum number of estimators (ranging from 50 to 500 with step as 50). Learning Rate is chosen in log space to better investigate the non-linear effect this parameter has on prediction accuracy. The maximum estimator size is set to 500, about one-fourth of the raw pixel feature size, which we assumed to be large enough to learn the essence of the sub-dataset.

Table 6: Hyperparameter Tuning of Adaptive Boosting Algorithm

# Estimators	50	100	150	200	250	300	350	400	450	500
Learning Rate = 0.001	0.220	0.231	0.244	0.245	0.245	0.249	0.248	0.248	0.249	0.248
Learning Rate = 0.01	0.248	0.269	0.282	0.289	0.292	0.298	0.300	0.308	0.307	0.310
Learning Rate = 0.1	0.314	0.321	0.33	0.334	0.337	0.338	0.331	0.338	0.338	**0.344**
Learning Rate = 1.0	0.298	0.304	0.317	0.306	0.297	0.300	0.294	0.299	0.298	0.296
Learning Rate = 10	0.146	0.142	0.153	0.157	0.154	0.159	0.16	0.158	0.153	0.153

From the Table 6, it turns out that accuracy has very limited growth or change when estimators size exceeds 300, for Learning Rate 0.001, 1.0 and 10.0 the accuracy even reaches its maximum when estimators size is 300. This gives us the hint that the number of weak learners in ensemble may not always contribute positively to the overall learning performance, but it does has an increasing trend when estimator size is small that makes a good impetus to the accuracy. On the other hand, through inspecting all the columns, we found out that learning rate has similar effect on accuracy regardless of estimator size. The maximum accuracy of the same amount of estimators basically occurs around learning rate being 0.1, and learning rates much less or greater than 0.1 show a decreasing trend bi-directionally. This finding makes us focus more on estimator size when learning rate being 0.1, and it basically shows an increasing trend as estimator size grows, and reaches the maximum when having 500 estimators.

Given that Learning Rate = 0.1 and Estimator size = 500 provides the best result, we hereby use this setting for the following task. Confusion matrix is a common measure of performance of classification model. The entities on the main diagonal of the matrix represents the correctly predicted samples number. After normalizing, those entities can represent We found some interesting things from confusion matrices of all trained algorithms.

4.4 Resampling for Validation

In the experiment, we use two resampling techniques for validation: k-fold cross-validation and boostrapping.

k-fold Cross Validation. We choose $k = 10$ for k-fold cross validation, which means the size of the testing set is $3500/k = 350$, and the size of the training set is $3500 - 350 = 3150$. In order to keep the dataset balanced, we extract 450 and 50 images from each emotion to form the training and testing set respectively.

Table 7 shows the result of 3 different algorithms, perceptron, adaboost, and SVM. To be specific, the accuracy is calculated according to the formula below:

$$accuracy = \frac{\#\ of\ images\ classified\ correctly}{\#\ of\ total\ test\ images}$$

Table 7: Accuracy of the algorithms with cross validation.

Perceptron	**Adaboost**	**SVM**
0.2337	0.2557	0.3323

Table 7 shows SVM achieved better performance than perceptron and adaboost. The most possible reason is we used linear model for perceptron and adaboost, but used radial basis function kernel for SVM, which may better capture the features of emotion images. Hyperparameters of different models will be tuned in our future work.

We also plot the confusion matrices of the three algorithms in Fig. 5 (a)-(c). From this figure, we could see emotions such as "surprise" and "happy" can be recognized more accurately, while emotions like "angry" and "surprise" are hard to be classified. The detailed reasons of accuracy gaps between different emotions will be explored in the final report.

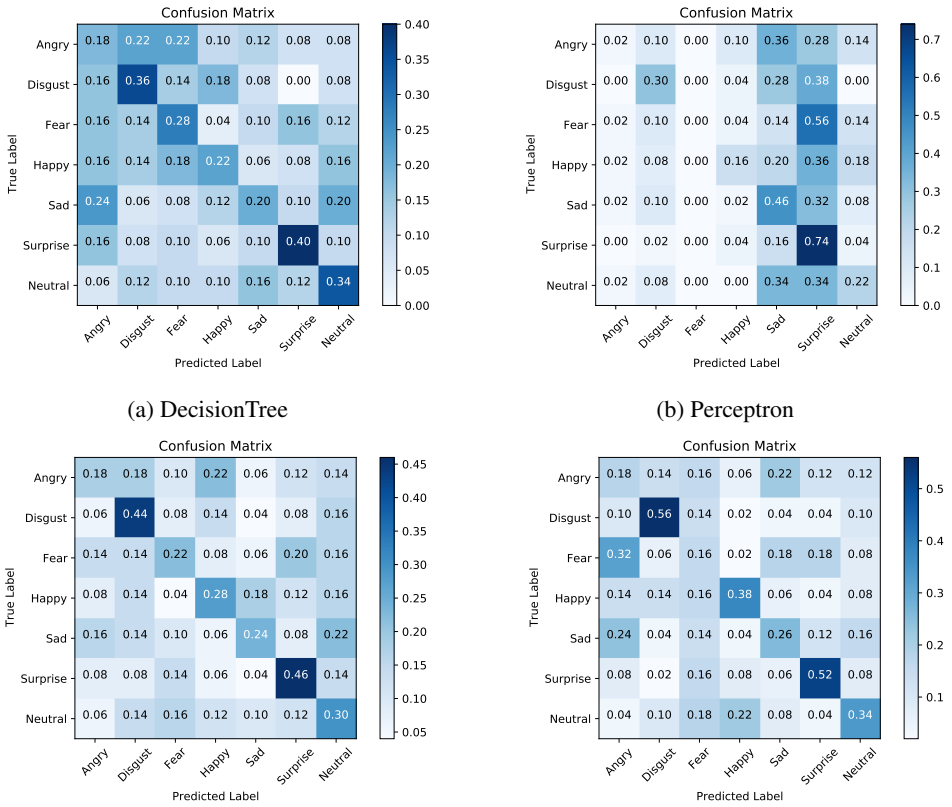
Bootstrapping. In bootstrapping, images of different emotions are sampled with replacement. In our experiment, we choose $B = 10$ and calculate the average classification accuracy of all the bootstrapping processes. Table 8 shows SVM achieved better performance than perceptron and adaboost in bootstrapping. We also plot the confusion matrices of the three algorithms in Fig. 5: (d)-(f), which are similar to the results of k-fold cross validation.

4.5 Evaluation Results

ROC curves are typically used in binary classification to study the output of a classifier. By varying the offset for a classifier (e.g., SVMS, logistic regression), we can get different sensitivity (TP/(TP+FN)) and specificity (TN/(TN+FP)). In this report we use the complementary part of specificity, i.e. false positive rate (fpr) as the x axis, and true positive rate (tpr, sensitivity) as the y axis, thus getting curves starting from (0, 0) and ending up to (1,1). AUC (Area Under Curve, < 1, usually > 0.5) denotes the area under the ROC curve, represents the performance of the classifier. The larger the AUC, the better the classifier.

In order to extend ROC curve and ROC area to multi-label classification, we binarized the multi-label output. One ROC curve can be drawn per label, but here we use macro-averaging, which gives equal weight to the classification of each label. Besides, each algorithm was run by 10-fold cross validation with the best tuned hyperparameters.

It can be shown from the ROC curves that SVM has the best prediction performance over the rest algorithms, Perceptron and AdaBoost have very similar performance, and DecisionTree has the worst



(a) DecisionTree

(b) Perceptron

(c) AdaBoost

(d) SVM

Figure 5: Confusion matrices of different algorithms under k-fold cross validation.

accuracy, and they are all better than random guessing (AUC > 0.5). Specifically, DecisionTree algorithm becomes much nearer to 0.5 curve as fpr grows, meaning that this model will not predict correctly much more even if the offset (threshold) above which is positive gets lowered down.

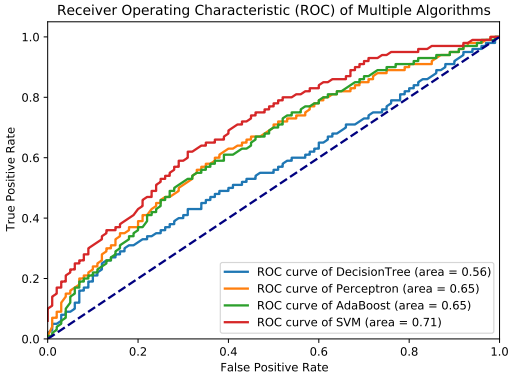


Figure 6: Receiver Operating Characteristic (ROC) Curve

5 Analysis

In this section, we analyze the experimental results, both qualitatively and quantitatively, and further discuss our findings from the experiments. Specifically,

Table 8: Accuracy of the algorithms with bootstrapping.

Perceptron	**Adaboost**	**SVM**
0.23142503	0.24360149	0.31108016

5.1 Image Encoding: Raw Pixels versus Combining Geometric Features

In section 4.2, we mentioned two image encoding methods, which are raw pixels encoding and facial landmarks encoding. In this subsection, we will use the idea of hypothesis testing to compare the above two encoding methods. Since the size of each image is $48 \times 48 = 2304$, only using 68 facial landmarks will lose important information of human emotions. Therefore, we use raw pixels encoding plus facial landmarks instead.

Under the best parameters tuned in section 4.3, we compare the average accuracies of 4 algorithms, including perceptron, adaboost, SVM and decision tree on 10-fold cross validation. Results are shown below in Table 9.

Table 9: Comparison between accuracies on 4 algorithms under different encoding methods.

	Perceptron	Adaboost	SVM	Decision Tree
Raw pixels	0.244	0.293	0.349	0.228
Raw pixels + facial landmarks	0.272	0.344	0.437	0.276

Although it is obvious that the second encoding method which uses raw pixels plus facial landmarks achieves better performance on all 4 algorithms. It is more rigorous to use a hypothesis testing method to prove the second method is better than the first one.

Here we use Wilcoxon signed-rank test, which tests the null hypothesis that two related paired samples come from the same distribution. In particular, it tests whether the distribution of the differences x - y is symmetric about zero (two-sided) or greater than zero (one-sided). For comparison, we run each algorithm 10 times on each encoding method, and get 10 accuracy differences. Then after running the 4 algorithms, we get 40 difference values and put them together to do the Wilcoxon signed-rank test using

```
w, p = wilcoxon(d, alternative='greater')
```

where p represents the significance level. Here we got $p = 3.44e - 07$, which means

5.2 Comparison between Cross-Validation and Bootstrapping

5.3 Numbers of Samples Versus Accuracy

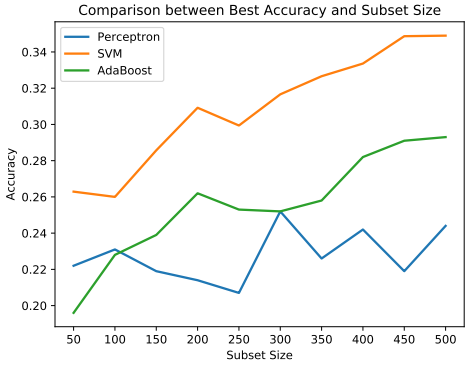
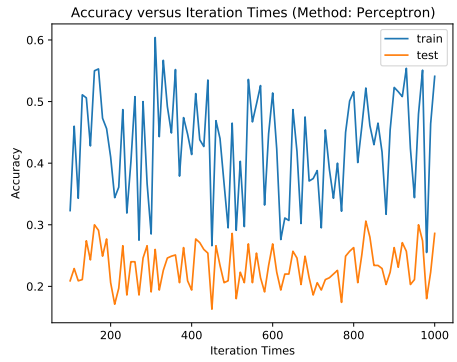
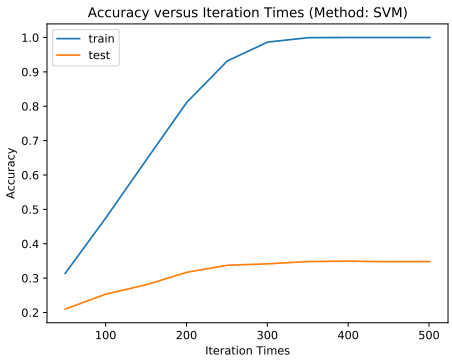


Figure 7: Accuracy versus Sample Size

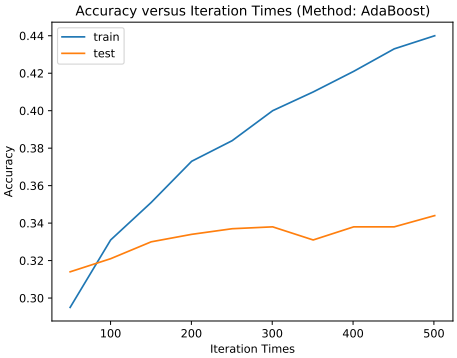
5.4 Numbers of Iterations Versus Accuracy



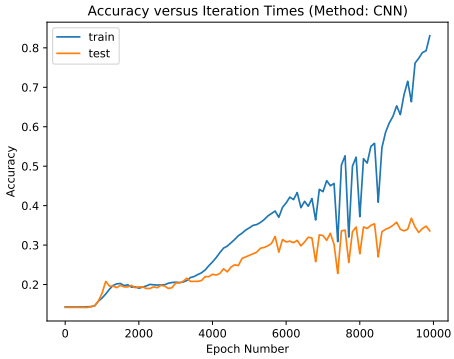
(a) Perceptron [Rosenblatt, 1957]



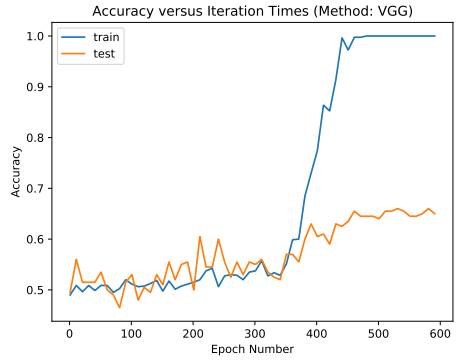
(b) SVM [Cortes and Vapnik, 1995]



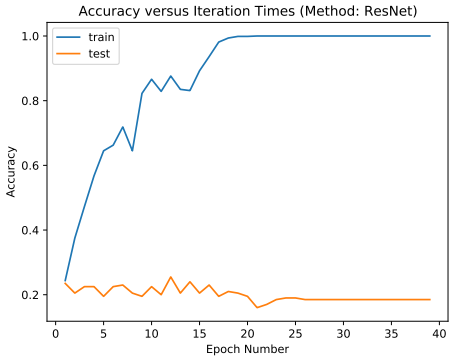
(c) AdaBoost [Freund et al., 1996]



(d) ConvNet with 5 layers [LeCun et al., 1995]



(e) VGG-16 [Simonyan and Zisserman, 2014]



(f) ResNet-34 [He et al., 2016]

Figure 8: Accuracy versus numbers of iterations (epoches).

In Fig. 8 we compare the accuracy of all of the six discussed algorithms versus their number of iterations during training. We found the ResNet-34 (Fig. 8f) converges quickly and only after a few iteration it achieves 100% accuracy in the training set; however, it performs poorly in the test set, showcasing the overfitting phenomenon and a lack of generalizability. In comparison, the simple CNN (Fig. 8d) converges slowly but generalize better. All of figures show that the algorithm in training sets outperform theirselves in test sets, which align with our intuition. From the visual results, we can make a rudimentary conclusion that VGG-16 is the winner in this dataset but we need techniques discussed in Section 4.4 for further validation.

5.5 Comparison of Different Algorithms

5.6 Regularization Parameter Versus Accuracy

6 Conclusion and Future Work

In this work, we comprehensively reviewed four classic machine learning algorithms (i.e., Perceptron, Adaptive Boosting, and Support Vector Machine) on a subset of FER2013, fine-tuned their hyperparameters, and evaluated their performance using two resampling techniques for validation (i.e., 10-fold cross-validation and boostrapping with 10 repetitions). We found that SVM with radial basis kernel outperformed the other three; while the Perceptron algorithm, restricted by its linear nature and expressiveness, is the weakest among the four. We also tried two ways to construct the feature vector from a raw input images: raw pixels and facial landmarks, and concluded that raw pixel incorporated with hand-crafted geometric features (i.e., distances of selected semantic features calculated from the facial landmarks) achieved the best prediction.

In comparison with deep learning-based methods, although VGG-16 outperforms these four methods by a large margin, these fine-tuned machine learning methods have strength in explainablity and generalizability, and less likely to overfit the dataset.

References

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Paul Ekman. Strong evidence for universals in facial expressions: a reply to russell's mistaken critique. 1994.

Paul Ekman and Wallace V Friesen. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, 17(2):124, 1971.

Irfan A. Essa and Alex Paul Pentland. Coding, analysis, interpretation, and recognition of facial expressions. *IEEE transactions on pattern analysis and machine intelligence*, 19(7):757–763, 1997.

Beat Fasel and Juergen Luettin. Automatic facial expression analysis: a survey. *Pattern recognition*, 36(1):259–275, 2003.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Ian J Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, et al. Challenges in representation learning: A report on three machine learning contests. In *International conference on neural information processing*, pages 117–124. Springer, 2013.

Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Fuzail Khan. Facial expression recognition using facial landmark detection and feature extraction via neural networks. *arXiv preprint arXiv:1812.04510*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Robert I Lerman and Shlomo Yitzhaki. A note on the calculation and interpretation of the gini index. *Economics Letters*, 15(3-4):363–368, 1984.

Shan Li and Weihong Deng. Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing*, 2020.

Maja Pantic and Leon J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12):1424–1445, 2000.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

Caifeng Shan, Shaogang Gong, and Peter W McOwan. Facial expression recognition based on local binary patterns: A comprehensive study. *Image and vision Computing*, 27(6):803–816, 2009.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27 (3):379–423, 1948.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Eric W Weisstein. Vector norm. *https://mathworld. wolfram. com/*, 2002.

Guoying Zhao and Matti Pietikainen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):915–928, 2007.

Ruicong Zhi, Markus Flierl, Qiuqi Ruan, and W Bastiaan Kleijn. Graph-preserving sparse nonnegative matrix factorization with application to facial expression recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):38–52, 2010.

Lin Zhong, Qingshan Liu, Peng Yang, Bo Liu, Junzhou Huang, and Dimitris N Metaxas. Learning active facial patches for expression analysis. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2562–2569. IEEE, 2012.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

A Program Manual and Running Environments

```
In this manual, we introduce the code structure and running environments.

Hardward Requirement
--------------------
* CPU
* GPU (at least 8GB memory and compatible with CUDA)
* RAM (at least 32GB)

Software Requirement
--------------------
* Unix-like System
* Python >= 3.5
* PyTorch >= 1.5.0
* OpenCV (Python binding only) >= 4.1.0
* dlib (Python binding only) >= 19.0.0
* sklearn >= 0.21.0
* NumPy >= 1.17.1
* matplotlib >= 3.2.0

Code Structure
--------------------
* Data Processing & Image Encoding:
    - FER2013.py
* Ploting:
    - Visualize.py
* Cross Validation & Boostrapping, Hyperparameter Tuning:
    - Evaluation.py
* Algorithm Implementation:
    - Perceptron.py
    - SVM.py
    - DecisionTree.py
    - AdaBoost.py
    - CNN.py
    - VGG.py
    - ResNet.py

Sample codes can be found at the end of each file.
```