Recognize Animation Characters with Machine Learning

Xin Guan, Ziqian Ge

Abstract

Please provide a brief abstract of your project.

1 Introduction

As the fandom community of animation grows, a large number of fan-arts made by non-official illustrators began to show up and gradually became an significant part of the community. Fan-arts spread on the internet are often not labeled or are hard for people, especially those who are new to the community, to recognize the illustrating character, since many anime characters shares similar characteristics and different illustrators may shift some of the features base on their own taste. Therefore, one character may look very different under different painters and many characters may look alike.

(a) Same Character by different illustrators (b) Similar characters

Figure 1: Various Looks of a Single Character and Similar but different Characters

Untagged illustrations would make some trouble for animation community members. Large number of 'Who is she/he' questions are emerging on social networks plantforms like Twitter, 2Chan and Bilibili. Even some video makers are making videos on answering those questions. On the other hand, within the machine learning community, if anyone is trying to make a illustration suggestion system based on the features or characteristics of anime characters, or trying to use GAN (Generative Adversarial Network) to generate fake anime illustrations, they might have to tag those pictures manually.

Automatic character recognizing would then make some differences when one is facing a large set of untagged illustrations or would like to add/optimize tags based on character, for example, a character may be bounded to specific tags, like blond, green-eye, sword, armored, etc.

This project is aiming to use machine learning techniques to automatically recognize characters in illustrations, based on their faces. We are mainly focusing on characters from japanese styled mangas and aminations. This project is making use of a variety of classification models including models like logistic regression classifier, random forest classifier and support vector machine and neural networks.

In order to make it possible to train non-neural-network models, we performed feature extraction techniques on images during the preprocessing stage. The first strategy is extracting the texture, color and

shape of the image contents and then flattern those features to an vector for the models to lean. The second strategy is extracting features using the output of the last layer of pre-trained ResNet[] as the image features and train them on non-neural-network models.

In this project, we did hyperparameter tuning on a rather small dataset that is collected manually by us and then train the model on the whole dataset which consists of ours and Nagadomi's anime face character dataset[]. Neural network performed best with a recognition rate of 90%. Support vector machine was the best model among non-neural-network models with ResNet extracted features. However, our first strategy leads to rather poor accuracy rate on all models (about 20% accurate rate on 178 classes).

2 Technical Approach

In this project, we made use of techniques relating to image feature extractions, classification models and neural networks to tackle the animation image classification problem.

1. **Feature Extraction Algorithms**
   Initially, we naively flatterned the RGB channels of a image to a vector as the input of the models, but find out that about 5-15 minutes are required to train a model even on the small dataset collected by ourselves. Additionaly, the prediction is not accurate since the flatterning process destroies the spacial relation between each pixels. Therefore, we find some way to extract the features of a image.
   - **Image Moment**
     Image moment is a certain particular weighted average of the image pixels' intensities. A uniqueness theorem [8] claim that a piecewise continuous 2D function $f(x, y)$ with nonzero values in a finite part of the xy plane, then moments of all orders exist and moment sequence is uniquely determined by $f(x, y)$. Image moments are useful to describe the shape of a image content. In this project, we made use of pre-implemented function $hu\_moment()$ in `OpenCV` [1] to process the image.
   - **Haralic Texture**
     Haralic texture, proposed by R. M. Haralick et al. in 1973, is describing a textural features "based on graytone spatial dependancies" [6]. This algorithm is trying to conclude the texture of a surface and the "structural arrangement of surfaces and their relationship to the surrounding environment"[]. In this project, we made use of the algorithm to extract the texture information of the images. The extraction of Haralic Textures is implemented in python package `mahotas` [3], with function $mahotas.features.haralick()$
   - **Histogram of Colors**
     In order to zip the information of color of a image, we decide to calculate how color are ditributed in HSV channels. To discretize the continuous distribution of colors, we decide to use 8 bins in each channel and thus store the histogram of colors to a vector. Again, we made use of `OpenCV` [1] to process the image.
2. **Prediction Models**
   In this project, we made use of many of the models that we learned in this course as well as some models in the prerequisite of this course. The implementation of these models are from python package `scikit-learn` [11].
   - **Logistic Regression**
     Logistic regression is used to model the probability of a certain class. With the feature input $x \in \mathbf{R}^n$, calculate $w^T x + b, w \in \mathbf{R}^n, b \in \mathbf{R}$ the probability of x in certain class is denoted by

     $$p(x = X|w) = \frac{1}{1 + e^{-(w^T x + b)}}$$

     In order to tacle overfitting problem, we also add a regularization strength λ:

     $$p(x = X|w) = \frac{1}{1 + e^{-(w^T x + b)}} + \lambda ||w||_2^2$$

     where $||w||_2^2$ is the l_2-norm of w. λ is a hyperparameter need to be determined before training. Training logistic regression is through gradient descent.

In `scikit-learn` [11], this problem is formed as a optimization problem:

$$\min_{w,c} f(w) + c\sum_{i=1}^{n} \frac{1}{1+e^{-y_i(w^T x_i+b)}}$$

where $f(w) = ||w||_2^2$ if we choose l_2 penalty.

In order to enable the model to classify multi-labels data, we train k models for k labels. Each model is a "one-vs-rest" classifier. When testing a given input, just find out the highest probability of all models and assign its class to the input.

- **Linear Discriminant Analysis**
  Linear discriminat analysis(LDA) is a Fisher's linear discriminant [5]. It is closely related to analysis of variance (ANOVA) and regression analysis. LDA assumes that the conditional probability density functions are normally distributed and the class covariances are identical. In this project, we uses multiclass LDA. This multiclass generalization is due to C. R. Rao [12]. Also, we directly use the implemented model from `Scikit-learn` [11]
- **k-nearest neighbors**
  k-nearest neighbors(kNN) is a non-parametric passive model used for classification. Upon an input object, the model classify by a vote of its neighbors. The most often appeared class in its k nearest neighbors are the output class. k is a hyperparameter that we need to decide before the training.
- **Decision Trees**
  Decision Trees are a non-parametric supervised learning method used for classification. The model is in a tree structures where its leaves represent class labels and branches represent conjunctions of features that lead to those classes. In this project we directly use the Classification and Regression Tree (CART) implemented in `scikit-learn` [11].
- **Random Forest**
  Random Forest is constructing a mutiple decisions trees at training time and outputting the class that is the mode of the classes of individual trees. This method is first proposed by Tin Kam Ho in 1995 [7]. Training process, called bagging [2], is repeatly select a random sample with replacement of the training set and fits a decision tree with these samples. We made use of the $RandomForestClassifier$ of `scikit-learn` [11] to perform the classification job.
- **Gaussian Naive Bayes**
  Naive Bayes is based on applying Bayes theorem with assumptions of features are mutually independent with each other. With Bayes rule, probability of a input vector x being class y_i is

$$P_{y_i}(x) = p(Y = y|X = [x_1, x_2, \dots]) = \frac{p(X = [x_1, x_2, \dots]|Y = y)p(Y = y)}{p(X = [x_1, x_2, \dots])}$$

  where Y is the class, x_i's are different features. With naive bayes assumptions, $p(X = [x_1, x_2, \dots]|Y = y) = \prod_i p(X_i = x_i|Y = y)$. Therefore,

$$P_{y_i}(x) \propto \prod_i p(X_i = x_i|Y = y)p(Y = y)$$

  In N class classification, we can calculate the value of all $P_{y_i}(x)$ for $i \in 1, 2 \dots N$ according to data set. Then, we find the largest $P_{y_i}(x)$ and assign y_i to x.

  Gaussian Naive Bayes is just making the assumption that the features are normally distributed. Given a dataset D with L data entries:

$$P(X_i = x_i|Y = y) = N(\mu_{x_i}, \sigma_{x_i}^2)$$

$$\mu_{x_i} = \frac{\sum_{k=1}^{L} X_{ki}}{L}$$

$$\sigma_{x_i} = \frac{\sum_{k=1}^{L} (X_{Ki} - \mu_{x_i})^2}{L}$$

  Where $N(\mu, \sigma)$ denotes normal distribution of mean μ, standard deviation σ^2.

- **Support Vector Machines**
  Support Vector Machines(SVMs) are supervised learning models used for regression and classification problems. With the assumption that the data is seperatable with a hyperplane, SVM is trying to find the hyperplane with the maximized margin between the data point in each class and the hyperplane. [13]

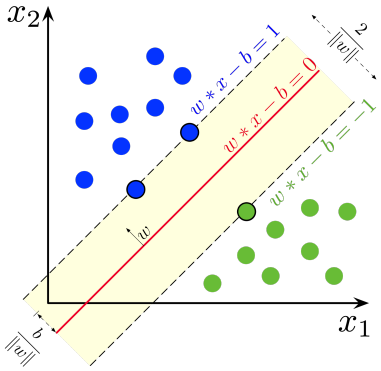


Figure 2: Maximum-margin hyperplane for a 2-class SVM

We made use of the C-Support Vector Classification (SVC) provided by `scikit-learn` [11]. We selected the "one-vs-one" scheme during the training. Hyperparameters need selection: regularization parameter C, kernel funciton, and the degree of polynomial kernel(if polynomial function is selected as kernel).

3. **Neural Networks**
   Neural Networks(NN), first introduced by Warren McCulloch and Walter Pitts [9] are computing structures inspired by biological neural networks. This system is composed of several layers of artifical neurons that mimics the firing of biological neurons.

   In this project, we used `pytorch` [10] to form and train neural networks. Particularly, we borrowed the ResNet [14] that is pre-trained on ImageNet [4] and added a layer of neurons representing our class labels and trained the net with our dataset.

3 Experimental Results

In this project, we have gone through a working pipeline shown in Figure 3. Firstly, we manually cleaned the Nagadomi's Anime Face Character Dataset and generated our own dataset with better image quality and modern animation characters. Then, we performed image pre-processing and feature extractions to generate three union of data: color-texture-shape information, ResNet-processed information and normalized RGB information. Then we designed different models according to these unions of data. According to our models, we did hyperparameter tuning on the self-collected dataset and then trained the model with the optimized hyperparameter on the whole dataset and evaluate the prediction result. Following are the detailed description of each step in the working pipeline.

1. **Dataset Overview**
   - **Nagadomi's Anime Face Character Dataset**
     Nagadomi's dataset contains 173 different animation characters. Most of the classes contains more than 50 images. However, it was collected 6 years ago, there are some problems in the dataset. Many of the characters are out of date so that fan art on them are no longer active. Also, most of the image size is around 100 * 100 pixels which are considered low resolution in modern standard. Therefore, some of them are difficult even for a senior community member to recognize. Furthermore, there exists some falsely labeled images, For example, shown in Figure 4, many pictures of Meirin Ri was labeled as Daidouji Tomoyo. Lastly, the data is in baised in some extend i.e. some of the classes do not contain sufficient amount of data, especially those non-popular characters. Therefore, we did a manually filtering process to delete bad images

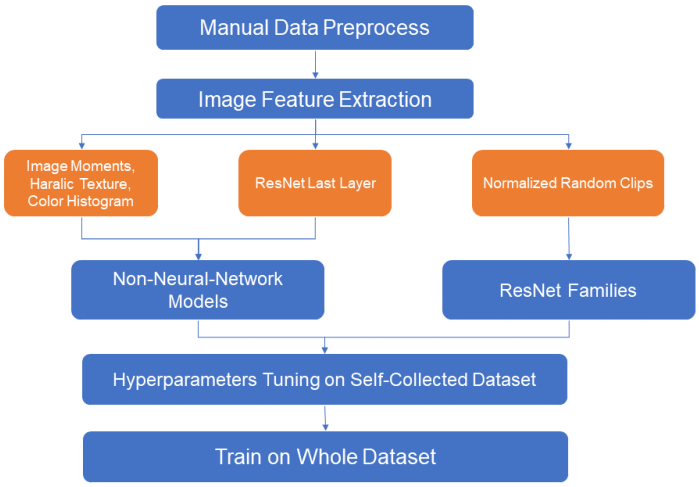


Figure 3: Working Pipeline

like ones shown in Figure 4. A copy of filtered data is stored separately.

(a) Meirin Ri labeled as Daidouji Tomoyo (b) Real Daidouji Tomoyo

Figure 4: The character on the left, Meirin Ri is falsely labeled as Daidouji Tomoyo

- **Self-collected Dataset**
  Due to the drawbacks of the existing dataset, we also collected some images with better quality. Totally, we collected images of 7 characters, each with over 100 images, manually from animations and illustrations published on web. These images are in similar format as images in Nagadomi's dataset, square-like images of faces of anime characters but with higher resolution (about 400 * 400) and better quality (high quality fan-arts).
- **Subset-ing the whole dataset**
  After the manual filtering process, there remains 163 characters. Since it was impracticable to train our models and tune hyperparameters using all of the data due to lack of computing power and time, we decide to manually pick a subset of characters from the whole dataset, each with over 100 images, to tune hyper-parameters. The training and testing on the whole dataset was performed on the tuned hyperparameter and the results will be reported. See Appendixes for the number of images in each class.

2. **Non-Neural-Network Models with Image Features inputs**

- **Data Pre-processing and Feature Extraction**
  This feature extraction process was a direct combination of HuMoments (implemented in OpenCV), Haralick features (implemented in mahotas), and color histogram (implemented in OpenCV). The result of the previous feature extraction was plugged as input into the next feature extraction method. The order of feature extraction we chose was: HuMoments → Haralick → color histogram.
- **Model implementation**
  We have trained logistic regression(LR) classifier, linear discriminant analysis(LDA) classifier, K neareast neighbors(KNN) classifier, decision tree(CART) classifier, random forest(RF) classifier, Gaussian Naive Bayes(NB) classifier, and support vector machine(SVM) classifier all using `scikit-learn` [11] to tune hyper-parameters on the subset described above.
- **Hyperparameter Tuning** Hyperparameters was tuned using the subset of dataset described above and the result using all of our data would also be produced after hyperparameters was tuned.
  (a) Logistic Regression classifier
  Since we chose to use "lbfgs" solver in scikit-learn, we can only use l2-norm in penalization, or use no regularization. Logistic regression models with or without regularization functions and with different regularization strength used in penalty function (C in scikit-learn) was trained to find out whether we are regularizing and the value of regularization strength. The regularization strength, C, was tested with floats ranging from 0.1 to 10.0 to find the best value.
  Testing results:

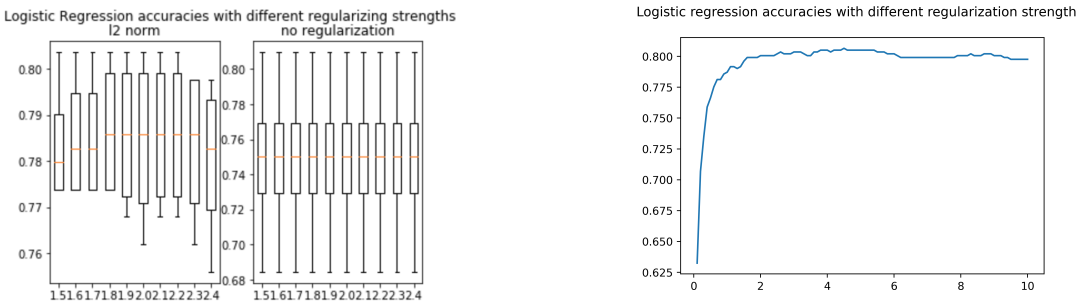


Figure 5: Logistic Regression classifiers hyperparameter tuning

  As shown in the result, we should have l2-norm regularization in penalty function, and the best regularization strength $C = 4.5$, with accuracy $= 0.807$

  (b) K Nearest Neighbors classifier
  We tried out different values of k, ranging from 2 to 30 to find the best value of k.
  Testing results:
  As shown in the graph, the k performing the best is $k = 6$.

  (c) Random Forest classifier
  We have tried forests with different loss function, Gini impurity function and Information Entropy function, both accuracy and computing time was record:
  As shown in the graph, we should use Gini impurity function as our loss function and the best number of estimators is 3000.

  (d) Support Vector Machine classifier
  We have tried different kernels, linear, polynomial, rbf, and sigmoid kernels to find the best kernel among them.

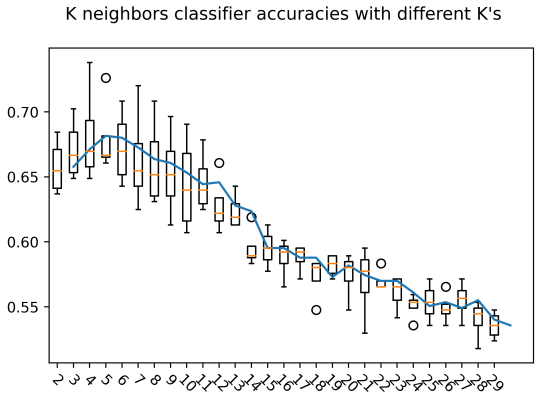
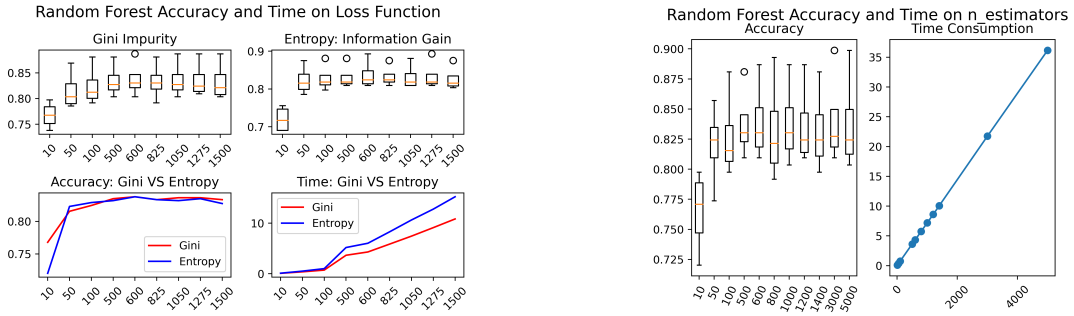
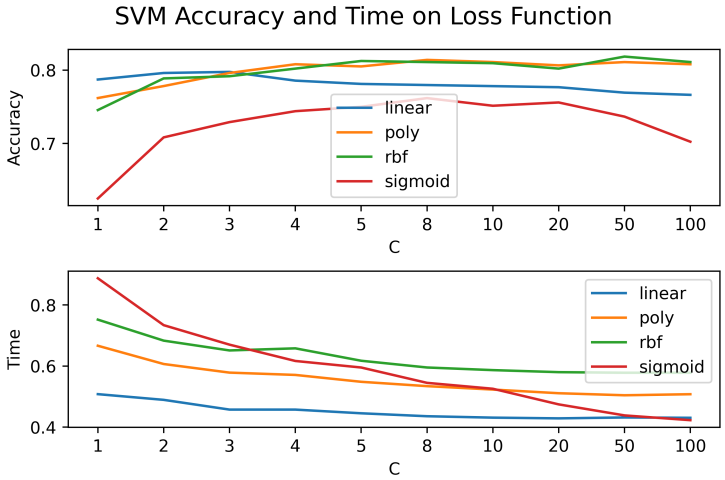
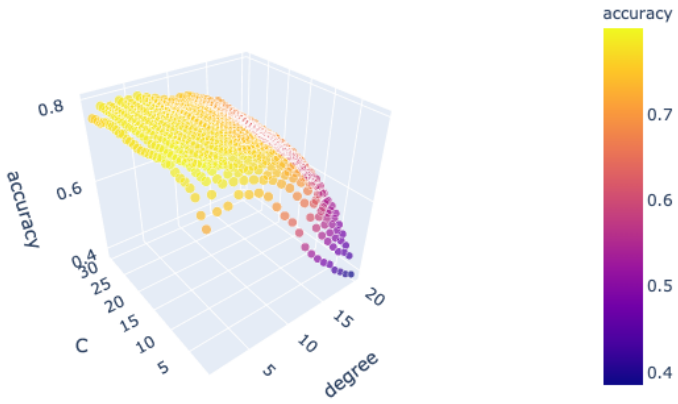


Figure 6: K Neighbor classifiers hyperparameter tuning





For polynomial kernel, we have also tried and tested different degree of polynomial kernel, ranging from 1 to 20, to find the best degree. The regularization strength, C, was also tested with floats ranging from 1 to 30.0 to find the best value.



The best hyperparameter $C = 4.0$ and $degree = 4$, with accuracy $= 0.799$
Here is a summary of the hyper-parameters:

LR	C = 4.5, penalty = l_2
KNN	$k = 6$
RT	Loss Function = gini, n_estimators = 3000
SVM	Kernel = poly, degree = 3, C = 11.0

- **Prediction Results**

Table 1: Non-Neural-Network Approaches' Accuracies

	self-collected			Whole dataset		
model	train	test	time	train	test	time
LR	0.949	0.756	$< 1s$	0.455	0.339	$37s$
LDA	0.987	0.420	$< 1s$	0.421	0.282	$1s$
KNN	0.784	0.563	$< 1s$	0.453	0.269	$1m\ 55s$
CART	1.0	0.605	$< 1s$	0.998	0.222	$9s$
RF	1.0	0.748	$3s$	0.998	0.549	$5m\ 11s$
NB	0.734	0.513	$< 1s$	0.394	0.134	$13s$
SVM	0.982	0.782	$< 1s$	0.661	0.393	$2m\ 15s$

As shown in the Table 1, the model performing the best on the whole dataset is Random Forest classifier.
When looking closely at the overall prediction results , we noticed that all models have some pattern making mistakes. One example is making mistakes distinguishing characters with similar hair color. Digging into the results with self-collected data, we found that models were easy to confuse characters with similar hair color. In the self-collected data set, all models made some mistakes distinguishing between `violet_evergarden` and `saber_arutoria_pendoragon` since they all have blonde hair. About 15% in average of `violet_evergarden`'s images were falsely recognized as `saber_arutoria_pendoragon`. See the examples of them: Figure 13.

3. **Non-Neural-Network Models With ResNet Last Layer Inputs**

- **Data Pre-processing and Feature Extraction**
  In this part, we directly used ResNet18, pretrained on ImageNet, as the image feature extractor. We first performed a random clip along with color normalization using the color average and standard deviation of the dataset. Then, we feed all images to the ResNet18 and use the activation values as the input. The input has a length of 1000. Before training, we also did a normalization on all vectors.
- **Model implementation**
  Same as the previous process, we trained a LR, LDA, KNN, CART, RF, NB, SVM classifiers.
- **Hyperparameter Tuning**

(a) violet_evergarden

(b) saber_arutoria_pendoragon

Figure 8: Characters with similar hair color may confuse the system

Hyperparameters was tuned using the subset of dataset described above and the result using all of our data would also be produced after hyperparameters was tuned.

(a) Logistic Regression classifier
Similar to the previous part, we iterate over several C to see which panelty is better generally. Then we find out its cooresponding regularization strength.
Testing results charts: Figure 8. As shown in the charts, we select $C = 1.1, penalty = l_2$

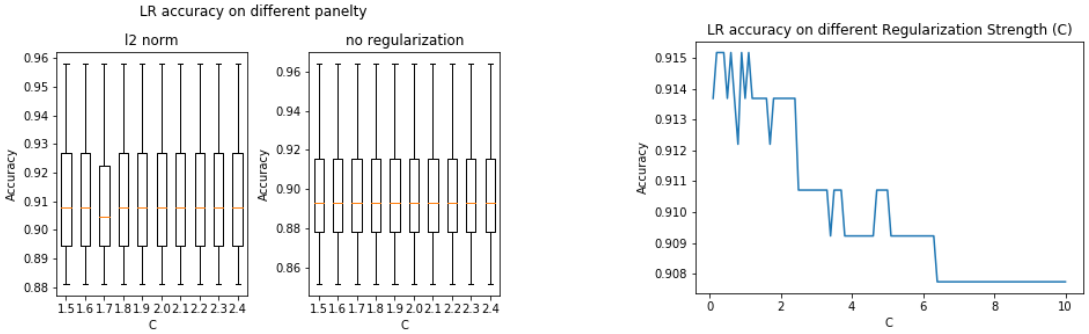


Figure 9: LR on ResNet Features hyperparameter tuning

(b) K Nearest Neighbors classifier
We tried out different values of k, ranging from 2 to 30 to find the best value of k.
Testing result charts: Figure 9. Therefore, the best k is 8.

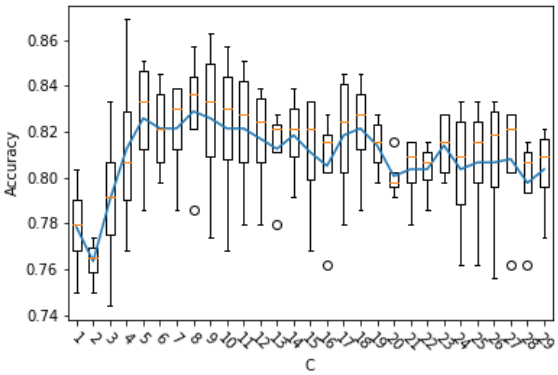


Figure 10: KNN on ResNet Features hyperparameter tuning

(c) Random Forest classifier
We have tried forests with different loss function, Gini impurity function and Information Entropy function, both accuracy and computing time was record. See Figure 10. As shown in the graph, we should use Gini impurity function as our loss function and the best number of estimators is 500.

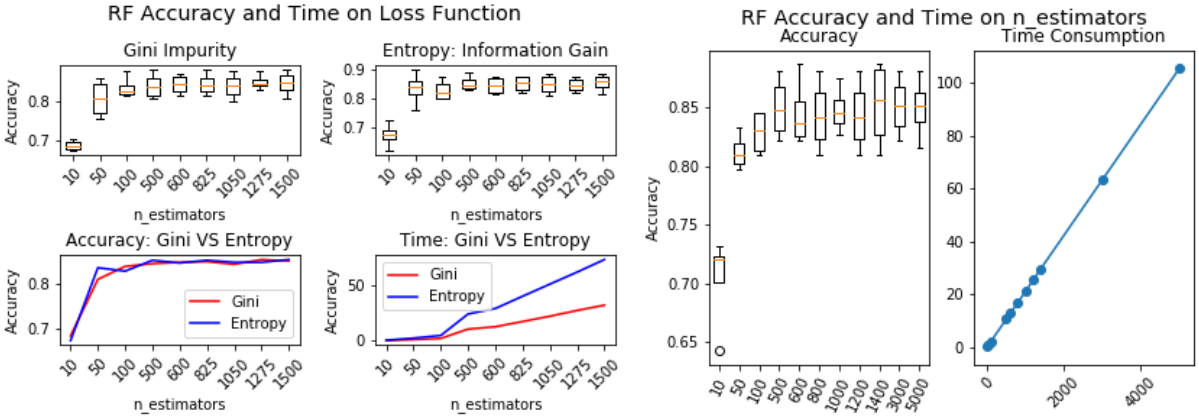


Figure 11: RF on ResNet Features hyperparameter tuning

(d) Support Vector Machine classifier
We have tried different kernels, linear, polynomial, rbf, and sigmoid kernels to find the best kernel among them. The polynomial kernel performs the best. See Figure 11. For polynomial kernel, we have also tried and tested different degree of polynomial kernel, ranging from 1 to 20, to find the best degree. The regularization strength, C, was also tested with floats ranging from 1 to 30.0 to find the best value. The best hyperparameter we have is $kernel = poly, C = 1.0$ and $degree = 4$.

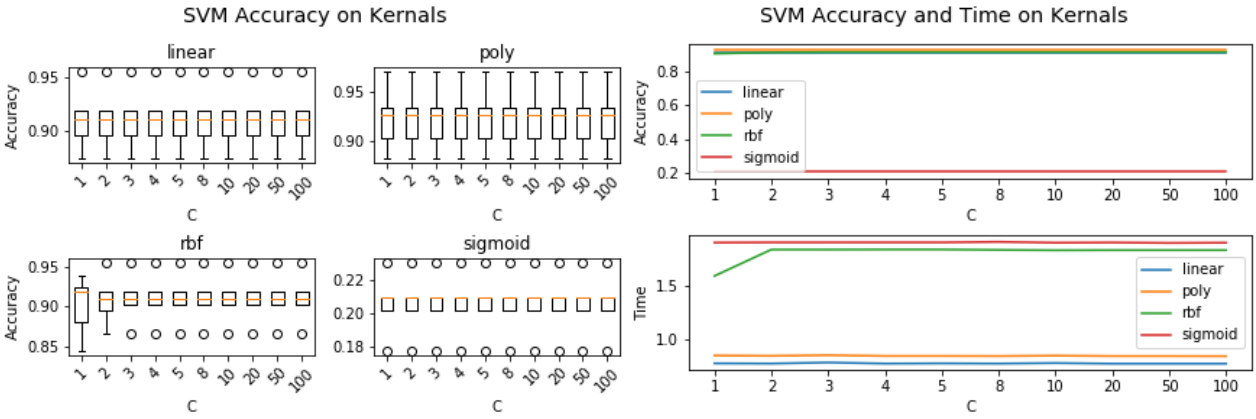


Figure 12: SVM on ResNet Features Kernels tuning

Here is a summary of the hyper-parameters:

LR	C = 1.1, penalty = l_2
KNN	$k = 8$
RT	Loss Function = gini, n_estimators = 500
SVM	Kernel = poly, degree = 4, C = 1

- **Prediction Result**
Compared to the previous method of feature extraction, features from ResNet is performing better. The pattern of making mistakes between characters with, for example, similar hair color, was still persisting. The severity was greatly ameliorated, however. The result on the whole dataset is also better than the previous approach. All the accuracy rate on test set

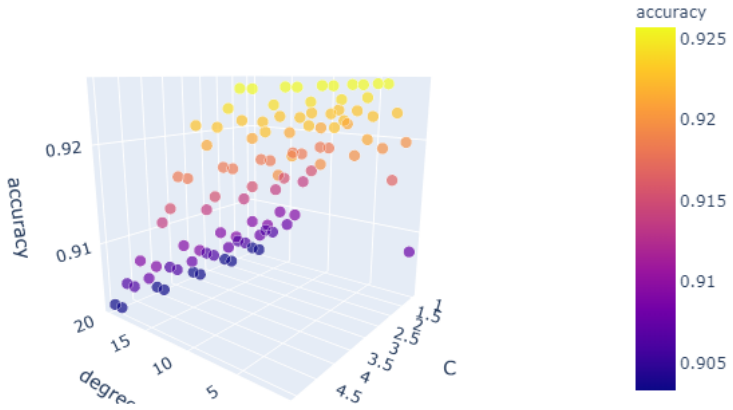


Figure 13: SVM on ResNet Features C tuning

decreases. The best three model is LR, SVM and LDA. see Table 1 for detailed accuracy rate.

Table 2: ResNet Feature Models Accuracy on Self-Collected Dataset

	self-collected			Whole dataset		
model	train	test	time	train	test	time
LR	1	0.941	1s	0.94	0.71	1m 10s
LDA	1	0.765	1s	0.915	0.751	2s
KNN	0.887	0.832	1s	0.609	0.414	3m 41s
CART	1	0.563	1s	0.998	0.126	1m 38s
RF	1	0.882	4s	0.998	0.51	13m 35s
NB	0.854	0.824	1s	0.603	0.482	25s
SVM	1	0.908	1s	0.998	0.679	3m 36s

4. **Neural Networks Approach (ResNet)**
   - **Data Pre-processing and Feature Extraction**
     Images was loaded using `torchvision` with initial transformations of:
     - randomly resizing into 224×224
     - randomly horizontal flipping
     - normalization using the mean and standard deviation of each RGB channel calculated from our whole dataset
   - **Model Implementation**
     Models were pretrained in PyTorch using ImageNet so we would be starting with a good enough weight to get potential prediction improvement. We used cross entropy function as our loss function and stochastic gradient descent as our optimizer with initial learning rate of 0.001, and momentum of 0.9. Learning rate was also adjusted by decaying 0.1 each 7 epochs, using learning rate scheduler packaged within `PyTorch`. Each model was trained for 25 epochs with training data and prediction results on both training and testing dataset were recorded after each epoch.
   - **Prediction results**

We tested ResNet of 18, 34, 50, 101 layers packaged within `PyTorch` using the subset of data mentioned above. The accuracy of ResNet of each layer are shown in the Table below:

Table 3: ResNet Accuracies on the subset

layers	train	test	time
18	0.965	0.969	$1m\ 19s$
34	0.970	0.987	$1m\ 44s$
50	0.964	0.975	$2m\ 33s$
101	0.965	0.975	$3m\ 59s$

The problem of distinguishing characters with similar features, e.g. same hair color, was persisting but was greatly ameliorated. Only about 12% of `violet_evergarden` was recognized as `saber_arutoria_pendoragon`

4 Participants Contribution

Data Collecting: Ziqian Ge found the Nagadomi's Anime Face Character Dataset. Xin Guan did the manual data cleaning. Ziqian Ge collected the `Illyasviel_von_einzbern` in the self-collected dataset; Xin Guan collected the rest.
Data preprocessing: Xin Guan did the data reforming in `data_cleaning.py` and completed the feature extractors of Non-Neural-Network data loaders(class `feature_extraction_dataloader` and `resnet_traditional_model` in `data_loader.py`). Ziqian Ge did the pytorch data loader and RGB information extractor (class `pytorch_dataloader` and function `img_stat` in `data_loader.py`) **Model Training and Hyper parameter Tunning**: Xin Guan finished the model structures for non-neural-network models (`resnet_traditional_models.ipynb` and `traditional_models.ipynb`); Ziqian Ge did the neural-network constructing and traning(`neural_network.ipynb`). Xin Guan tuned SVM and RT's parameters along with visualization; Ziqian Ge tuned LR and KNN's parameters along with visualization. **Project Writing**: Xin Guan Completed Introduction, Technical Approach and Non-Neural-Network Models With ResNet Last Layer Inputs in Experimental Results. Ziqian Ge finished Dataset Overview, Non-Neural-Network Models with Image Features inputs and Neural-Network Model in Technical Approach along with the Participants Contribution.

References

[1] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[2] Breiman, L. Bagging predictors. *Machine Learning 24* (1996), 123–140.

[3] Coelho, L. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software 1(1):e3* (2013).

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).

[5] Fisher, R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics 7* (1936), 179–188.

[6] Haralick, R. M. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics SMC-3* (1973), 610–621.

[7] Ho, T. K. Random decision forests. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, p. 278–282.

[8] Hu, M.-K. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory 8* (1962), 179–187.

[9] McCulloch, W. W. P. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–133.

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

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

[12] Rao, R. C. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society, Series B 10* (1948), 159–203.

[13] Wikipedia. Svm margin, 2020.

[14] Xie, S., Girshick, R., Dollar, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. pp. 5987–5995.