Assignment 2 - Group: 23

Tutors: Long Tan Le and Md Mashud Rana

Group members: Yihan Xu (yixu7841), Shengnan Wang (swan7185)

Assignment cover sheet

Personal Details of Students

Group Number	**Group 23**				
Family Name	**Given Name (s)**	**Student Number (SID)**	**Unikey**	**Contribution Percentage**	**Signature**
Xu	**Yihan**	**480143468**	**yixu7841**	**50%**	**Y.X**
Wang	**Shengnan**	**520160691**	**swan7185**	**50%**	**S.W**

Abstract

In this assignment, the classification dataset EMNIST-ByClass, a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset has been chosen[5]. Three classifiers with models have been built and applied to determine the best classifiers that classify images in this assignment. This report contains the theories behind different classifiers used in this experiment, the best parameter with predicted result of each algorithm as well as the final comparison for three classifiers to find out the best output result and determine the best model by utilizing the testing dataset. Moreover, this report provides instructions on how to run the code and describes the running time, the meaningful comments on the results, the design choices, the configuration of the computer, and the environment of the hardware and software that are used for performance evaluations.

Introduction

MNIST dataset is one of the most popular balanced datasets in English handwritten digit and all the datasets are separated into training and testing sets[5]. EMNIST dataset is the extended version of MNIST dataset including 697932 training data and 116323 testing data, each image of the dataset is about 28x28 pixels. The sample images of the EMNIST dataset are shown in Fig. 1.

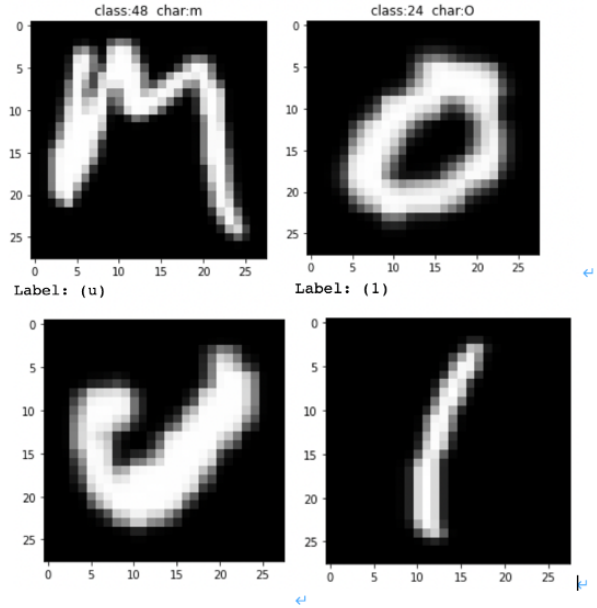


Figure 1 Sample images of the EMNIST dataset

The first step is using scipy.io.loadmat to load training and testing data sections in mat file from EMNIST-ByClass dataset.

The second step is preprocessing the dataset in order to transform features on one similar scale that helps to improve stability and the performance of the training model, a method Normalization has been applied when pre-process dataset. In addition, PCA, a dimensionality-reduction method that is regularly utilized to reduce the dimensionality of large data sets should be performed during the step of pre-processing the dataset because the original datasets are too big to run. Before applying classification algorithms, reducing the dimension of datasets could avoid high-volume computational data and long code running time.

The third step is implementing classification algorithms, tuning parameters, and comparing different classifiers. Multiple classifiers SVM, Random Forest and CNN have been utilized. In

every classifier, firstly, find the best parameters using random search validation according to the theories and methods of each algorithm. In terms of the top-1accuracy metric, after keep searching the performance of each classifier, it would eventually show the fine-tune hyperparameters and the running time along with the accuracy rate calculating in number of correct classifications /total number of test examples used $*$ 100%. Then compare the three classifiers used in the experiment with their own best hyperparameters, the accuracy and the running time together. After validating the performance on training sets, the best model and the specifications of the computer hardware and software used would both be evaluated among these three classifiers. Finally, make a conclusion of the best model and analyze the output of testing data.
The last step is outputting the results containing the algorithm and parameters that yield the best result from all the other algorithms. The best trained model generated by code along with code and report would be saved and submitted to Canvas.

Related Work or Previous work

In today's era of rapid development of big data and the Internet, deep learning and machine learning are becoming more and more widely known. In order to train the EMNIST dataset, this project uses a more lightweight NumPy library, sklearn library, which can be embedded in terminal devices such as servers and mobile phones. In addition, in order to improve the accuracy of EMNIST data in the environment, the pytorch library was also used to build a convolutional neural network from scratch.

Overview of NumPy Library used in this project:

Firstly, read and process the data. Then label the data and initialize the training data. Use the np.random. random function to return a set of random data with a standard normal distribution. At this point, it can be seen that the data returned by NumPy is between 0 and 1 randomly. Then update the parameters and use the matplotlib library to draw graphics, from which weights and biases can be obtained. Finally calculate and improve the accuracy that represents the effect of learning intuitively. By plotting the learning curve with matplotlib, it can be shown that the accuracy of the training set is basically the same as the accuracy of the test set, which means that there is no overfitting.

Overview of Pytorch Library used in this project:

Firstly, read the data using the Data Loader since Pytorch contains the EMNIST dataset that has been officially implemented and can be utilized directly. Once this data is ready, subsequent load commands will not re-download and the prepared data can be reused. The data is then preprocessed to define the model with some hyperparameters and a network that contains two convolutional layers. After outputting the dimensions are used to determine which number is recognized, then test the training data and adjust the hyperparameters and model through the training results if needed. Finally, use the .to method to move the network to the GPU after instantiating a network and defining the training function where all operations for training and testing data are encapsulated into, which reflects the benefits of encapsulation, as long as only a few lines of code need to be written.

Difference between Pytorch Library and TensorFlow Library:

With the development of this technology, many deep learning libraries continue to come out such as TensorFlow. But sometimes after using these libraries, people cannot pay attention to the underlying logic of deep learning. Moreover, the process occupies a huge amount of memory, and users cannot easily use python. According to the research, the previous work shows successful techniques utilizing TensorFlow on the similar datasets. Compared to Pytorch, all operations require writing a lot of boilerplate code for low-level operations when using the TensorFlow library. Over time, a number of advanced wrappers have gradually emerged to simplify the way of using TensorFlow. As a result, there is a lot of freedom in how to use TensorFlow, and how to choose the framework that best matches the task. In contrast, when building a CNN classifier that classifies handwritten digits, PyTorch looks like a programming framework, which provides useful abstractions in specific domains. There are many ready-to-use modules in the torchon package that can be utilized as the basis, like the module used to build custom CNN classifiers and the building blocks to create complex deep learning architectures. Both TensorFlow and PyTorch provide useful abstractions to reduce the amount of boilerplate code and speed up the deployment of models because both are about the same computational speed under the same conditions. The main difference between these two libraries is the static calculation graph and the dynamic calculation graph. Iit is very troublesome to migrate and debug data parameters between CPU and GPU for the static calculation graph in TensorFlow. However, in the dynamic calculation graph in pytorch, the migration of data parameters between CPU and GPU is very flexible and debugging is easy.
At present, we have achieved an accuracy of about 87% using CNN classifier, but it is still far from "intelligent". The recognition rate of humans can reach more than 98%, and the purpose of artificial intelligence is to simulate the human brain. So there is still a long way to go, and we still need to work hard to optimize.

Methodology

2.1 Load The Data

First load the data from the website, using mat file from EMNIST-ByClass dataset in MATLAB format dataset. The labels and data from train, test and map sets are extracted separately and used as data samples for training, testing, and drawing. Because the original data is unprocessed, we first rotate the image by -90 degree and then flip horizontally, then randomly shuffle the data, and take 10% of the dataset (1/10 for train and test respectively, 69793 train samples and 11632 test samples) as the samples for this project.
Since there is too much data in the EMNIST data set, our computer CPU capacities are limited and cannot run such a huge amount of data(excessive load during operation), so only a part of it is taken. Then Normalization and PCA are used for pre-processing the dataset.

2.2 Pre-processing

2.2.1 Normalization

Min-Max scaling Normalization is used for pre-preprocessing, all data are scaled according to the range (maximum-minimum value), meaning that the minimum and maximum value of a feature goes to be 0 and 1, respectively[7]

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Figure 2 Normalization formula[7]

Since the maximum value of image pixels is 255, all feature data are divided by 255 to scale all image pixels to 0-1.

2.2.2 PCA

PCA is a classic dimensionality reduction algorithm, which is a linear, unsupervised and global dimensionality reduction method. The principle of PCA is linear mapping. Project the high-dimensional space data to the low-dimensional space, retain the principal components, and the data in the principal components contain a large amount of information, but ignore the secondary information that is not important[3].

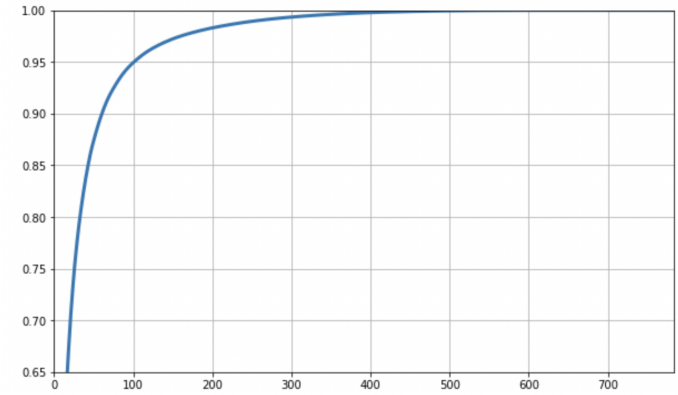


Figure 3 The variance occupied by each dimension for PCA

First draw an image to show the variance occupied by each dimension. The abscissa is the dimension, the ordinate is the percentage of data information that can be covered. The line chart shows that when the dimension is 150, more than 95% of the information is covered. So we reduce the dimension to 150, then train and convert.

Dimensionality reduction is useful for high-dimensional datasets. Although dimensionality reduction discards some data, discarding that part of information can increase the sampling density of samples. In addition, the eigenvector corresponding to the minimum eigenvalue of the dataset is often affected by noise. Discarding these data could reduce some noise[10].

2.3 Performance Metrics

2.3.1 Confusion Matrix

A confusion matrix is a matrix of size 2x2 which is used for binary classification. One axis represents actual values and the other is predicted values[6].

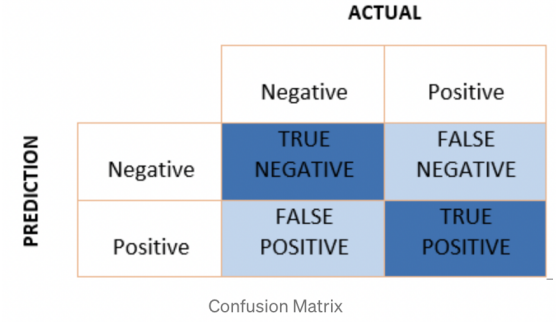


Figure 4 Confusion Matrix[6]

True Positive(TP) — model correctly predicts the positive class. The prediction value and actual value are both positive.
True Negative(TN) — model correctly predicts the negative class. The prediction value and actual value are both negative.
False Positive(FP) — model's prediction of negative class data is wrong, which means that the predicted value is positive, but the actual value is negative. FP is also called TYPE I error.
False Negative(FN) — model's prediction of positive class data is wrong, which means that the predicted value is negative, but the actual value is positive. FN is also called TYPE II error.

2.3.2 Accuracy
The accuracy is the percentage of correctly predicted results in the total original sample.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Although the accuracy rate refers to the overall correct rate, when samples are not balanced (the number of positive samples and the number of negative samples are much different), it may not be considered as a good metric to measure the results[4]. Therefore, in the case of imbalanced samples, the high accuracy obtained does not make any sense. We need to find new indicators to evaluate the model.

2.3.3 Precision
The accuracy rate focuses on the prediction result, it indicates how many of the predicted positive samples are exactly the real positive samples. So there are two possibilities when the prediction result is positive. One is to predict the positive class as a positive class (TP), and the other is to predict a negative class as a positive class (FP).

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The denominator is the number of samples which are predicted positive. The precision value lies between 0 and 1.

2.3.4 Recall
The recall rate focuses on the original sample, it indicates how many positive values in the sample are predicted correctly. There are two possibilities, one is to predict the original positive class as a positive class (TP), and the other is to predict the original positive class as a negative class (FN).

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The denominator is the number of all positive samples in the original dataset. Recall value lies between 0 and 1.

2.3.5 F1 score
F1 score is the harmonic mean of precision and recall, which takes into account both false positives in precision and false negatives in recall and provides a good performance on imbalanced datasets[6].

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

Figure 5 F1 score formula[6]

There is a weighted F1 score in which different weightage are assigned to recall and precision according to different situations[6].

$$F_\beta = (1 + \beta^2) * \frac{(Precision * Recall)}{(\beta^2 * Precision) + Recall}$$

Figure 6 Weighted F1 score formula[6]

Beta represents how many times recall is more important than precision. If the recall is twice as important as precision, the value of Beta is 2[6].

2.4 Models

2.4.1 Random Forest(RF)

2.4.1.1 Theory

A random forest classifier is a combination of tree classifiers. A random vector is independently sampled from the input vector, which is used for the generation of a tree classifier; every tree casts a unit vote, in order to select the most popular class to classify an input vector[7]. Random forests can be used for classification, where the combination of features at each node or randomly selected features are used for the generation of a tree[8]. The original random forest algorithm is a bagging method, which randomly selects observations of size B as a substitute, and then generates a sample of size B (called a bagging sample) from an initial dataset of size N[2]. All examples (pixels) used for bagging are classified according to the most popular class selected by combining the votes of all three predictors[2].

2.4.1.2 Parameter selection and implementation

After implementing PCA, the random forest can be tuned for better predictions with some hyperparameter tuning. So, more hyperparameter values can be considered starting from RandomSearchCV. Random search is used for searching, including random hyperparameter grid random search training, and evaluating random search. Call RandomizedSearchCV to generate random hyperparameters grids. For the search parameters, “n_iter” is set to 50 and “cv” is 5 for training to make a balance between performance and time.

When generating a "param_dist", its range of values applies to each hyperparameter. To instantiate RandomSearchCV, firstly, pass in the random forest model, then pass in "param_dist", the number of test iterations, and the number of cross-validations. The variables max_depth means the max number of levels in each decision tree while the “n_estimators” represents the number of trees in the forest and the “max_features” is the number of features at each split. The hyperparameter "n_jobs" determines how many processor cores should be used to run the model. Setting n_jobs will make the model run fastest because it uses all computer cores. Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster. Since max_depth ,n_estimators and n_iter have been chosen as variables, with the change of max_depth and n_estimators, after achieving the accuracy and the running time, then figure out how the relationship or the trend relates to these variables.

2.4.1.3 Method Ideas and Design Choices

To construct a random forest model, it is necessary to first determine the number of trees in the forest and determine the selection range of each parameter. Thereby a parameter dictionary is formed, and the model is used for adjustment. The larger the n_estimators is, the better the effect of the model has. However, the memory occupied, and the training and prediction time would also increase accordingly while the marginal benefit would decrease. Consequently, within the

affordable memory/time, the n_estimators should be selected as large as possible. Because using sklearn's RandomizedSearchCV could save time and avoid a lot of redundant searches, then the optimal parameter group under random parameters will be obtained from training the model using the parameters in the parameter dictionary. Take the best parameter group as the standard, and then take the values up and down to form a new parameter selection range and corresponding parameter dictionary. Using sklearn's GridSearchCV, after initially determining the approximate parameter range, it is beneficial to use carpet traversal to improve efficiency. Then use the parameters in the new parameter dictionary to train the model to get the final optimal parameters.

2.4.1.4 pros and cons

The main advantages of the random forest algorithm:

Because the random forest classifier uses the voting combination of all trees for classification, the classification accuracy of RF is higher than that of the basic simple classifiers. In addition, it can handle high-dimensional data and multicollinearity well. It is fast and insensitive to overfitting, showing the advantages of being fast and insensitive to overfitting[1].

One of the main advantages of random forests over other decision tree methods is that there is no pruning of fully grown trees. Trees are generated using feature combinations on new training data and are not pruned when reaching maximum depth[8]. Pal and Mather (2003a) showed that choosing pruning methods, rather than attributes, will affect the performance of tree-based classifiers[9]. Breiman (1999) showed that though RF does not prune the tree, the generalization error always converges[7].

The main disadvantages of the random forest algorithm:

Random forests have been shown to overfit on some noisy classification or regression problems. And for data with attributes with different values, attributes with more value divisions will have a greater impact on random forests, so the attribute weights produced by random forests on such data are not credible[12].

2.4.2 Support Vector Machines (SVM)

2.4.2.1Theory

SVM, a binary classification model defined by a classification hyperplane. Given a set of labeled training samples, the algorithm will output an optimal hyperplane to classify new samples (test samples).For a data set that is linearly separable in the original space, it is necessary to find a set of best deterministic segmentation hyperplanes so that the distance between the support vector and this plane is the largest, that is, the optimal segmentation hyperplane maximizes the training sample boundary. The support vector is also the sample point closest to the classification hyperplane. For datasets whose original space is nonlinear, a kernel function transformation is needed to map the original data sample points into the feature space. Then a linear hyperplane is found in the high-dimensional space, so that the hyperplane can achieve class division.

2.4.2.2 Parameter selection and implementation

Kernel functions are a method used to transform nonlinear problems into linear problems that can improve the Feature dimension of the model (low dimension to high dimension), so that SVM has better nonlinear fitting ability. The parameter choices in the kernel function type used in the algorithm are RBF, Linear, Poly, and Sigmoid. RBF is the radial basis kernel, which is the Gaussian kernel function. Linear refers to the linear kernel function while Poly refers to the polynomial kernel, and Sigmoid refers to the hyperbolic tangent function. When using SVM, if the kernel function is utilized, Feature Scaling (Normalization) must be performed on the Feature. In addition, if the training set m is too small, but the number of features n is large, the training data is not enough to fit a complex nonlinear model. In this case, only the linear kernel can be used, and the Gaussian kernel function cannot be used.

There are two most important parameters of SVM: C and σ in the kernel function. C is a penalty slack variable, similar to the role of λ in regularization. As the value of C is larger, the ability to fit nonlinearity is stronger. Large C represents High Variance, that is, the penalty for misclassification increases, which tends to be the case where the training set is completely divided correctly. As a result, the accuracy rate is high, but the generalization ability is weak when testing the training set. Small C stands for High Bias, the penalty for misclassification is reduced, so misclassification is allowed, and they are regarded as noise points. If the generalization ability is stronger, the value of σ is large and f is smoother. The Nonlinear performance is smaller means it is less sensitive to noise. Similarly, large σ represents High Bias, and small σ represents High Variance.

2.4.2.3 Method Ideas and Design Choices

Using SVM to choose a kernel function, which maps the data to a high-dimensional space through the function k(.,.).The support vector machine first completes the calculation in the low-dimensional space, then maps the input space to the high-dimensional feature space through the kernel function, and finally constructs the optimal separation hyperplane in the high-dimensional space. The kernel function can avoid directly computing in high-dimensional space when the result is equivalent.

In the assignment where is the case of dealing with non-linearity, SVM has been implemented using the sklearn library and sklearn.svm.SVC has been applied to set parameters. The training time of the SVC function increases with the square of the number of training samples. It is appropriate to use SVC here as the number of samples are not that large. First, import the data set and divide it into training and testing sets. Second, set candidate values for parameter matching then call the GridSearchCV method for parameter tuning and pass SVC(), tuned parameters, cv=5 into the function. Finally, train this classifier with the training set and call the best_params_ method to get the best parameter matching result directly. Then save and store the best model into the folder as requested.

2.4.2.4 pros and cons

The main advantages of the SVM algorithm:

It is very effective to solve the classification and regression problems of high-dimensional features, and it still performs well when the feature dimension is larger than the number of samples. When the sample size is not massive data, the classification accuracy is high, and the generalization ability is strong. Also, SVM only uses a subset of support vectors to make hyperplane decisions without relying on all data. Moreover, a large number of kernel functions could be utilized, which can be very flexible to solve various nonlinear classification and regression problems.

The main disadvantages of the SVM algorithm:

SVM is very sensitive to missing data, that is to say, if the feature dimension is much larger than the number of samples, the SVM will perform poorly. When the sample size of SVM is extremely large and the dimension of kernel function mapping is too high, the computational load would become large as well and it is not suitable to use SVM. In addition, it is difficult to choose a suitable kernel function when facing a nonlinear problem because there is no general criterion for the selection of the kernel function.

2.4.3 Convolutional Neural Network (CNN)

2.4.3.1Theory

Before the convolutional network comes out, the general network adopts the method of matrix multiplication, and each unit of the previous layer has an influence on each unit of the next layer. Reducing the dimension of a large number of parameters into a small number of parameters and then processing them is how CNN simplifies complex problems. Traditional convolutional neural

networks are obtained by stacking a series of convolutional layers and down sampling layers, but when they are stacked to a certain network depth, two problems will arise. (1) Gradient vanishing or gradient exploding (2) Degradation problem. Like other deep neural networks, convolutional neural networks (CNNs) are also composed of multi-layer network structures, which usually contain three types of network layers: convolutional layers, pooling layers, and fully connected layers. The convolution layer is generally a square grid of neurons. After the features are extracted through the convolution kernel, these features can be used for classification. CNNs often uses mini-batch gradient descent (Mini-Batch) to train the model and uses the sigmoid function as the activation function when it is a nonlinear transformation. When using ResNet in CNN, data preprocessing and the use of BN (Batch Normalization) layers in the network can speed up training to solve the problem of gradient disappearance or gradient explosion.

2.4.3.2 Parameter selection and implementation

- max_lr: maximum learning rate (initial learning rate)
- torch.optim.adam: The optimization function is torch.optim.adam. Adam designs independent adaptive learning rates for different parameters by computing the first moment estimates and the second moment estimates of the gradient.
- lr scheduler using StepLR, the gradient decreases proportionally after the specified step
- Gradient clipping is used, and the gradient is limited to a maximum of 0.1
- An 'epoch' means that when a complete dataset has been passed through the neural network once and returned once, this process called an 'epoch'
- Batch size: An epoch contains several batches. batch size is the Total number of training examples present in a single batch.

Passing the complete dataset once in the neural network is not enough, the 'epoch' needs to be set more in order to pass the complete dataset multiple times in the same neural network.[11]。 In this project, we set 'epoch' to 32.
Since one "epoch" is too large to input into the computer immediately, it needs to be split into several smaller batches. 'Batch size' is an important parameter, each training will take batch size samples in the training set for training. For a dataset, if the 'Batch Size' is too small, the data will be difficult to converge, resulting in underfitting. Properly increasing 'Batch Size' value will help improve memory utilization and running speed, but the memory capacity required will increase. Therefore, the purpose of tuning 'Batch Size' is to find a balance between memory efficiency and memory capacity[11].

2.4.3.3 Method Ideas and Design Choices
Resnet is a very common variant of CNN, because of its simple structure and excellent performance, it has been widely used Traditional convolutional networks or fully connected networks have more or less problems such as information loss and loss when information is transmitted. Meanwhile, it also causes the gradient to disappear or the gradient to explode, making the deep network unable to train. ResNet solves this problem to a certain extent. By directly passing the input information to the output, the integrity of the information is protected. The entire network only needs to learn the difference between the input and output, which simplifies the learning goals and difficulty.
Common resnet structures include Resnet18, Resnet50, etc. However, on the EMNIST dataset, because the task is simple and does not require such a complex structure, a simplified version of the 9-layer resnet is used. The structure of Resnetis shown in the figure 7.

The model consists of a preprocessing layer and two residual blocks. The preprocessing layer uses conv3 and max pooling to reduce the dimension and increase the channel dimension. The residual block is composed of two conv3 including skip connection, each conv layer is injected with bn. Finally the class of the picture is output through a classifier (full connection + SoftMax).

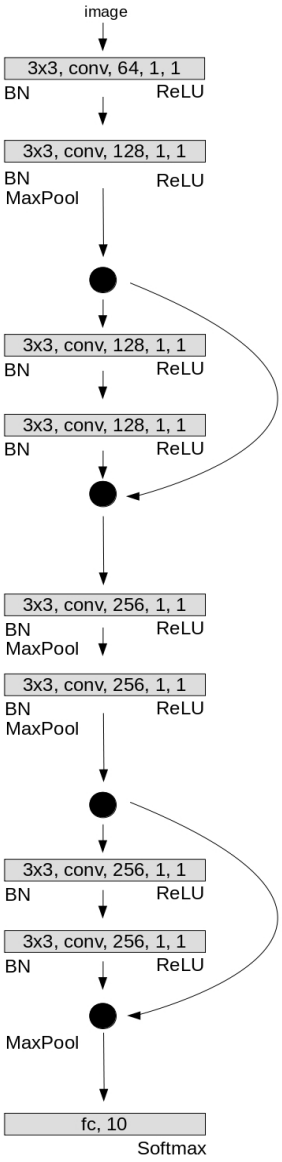


Figure 7 Architecture of CNN[12]

2.4.3.4 pros and cons

The main advantages of the CNN algorithm:

When the network is deeper, the number of parameters is controlled. The number of feature maps ensures the expressiveness of the output features due to the distinct layers. Instead of using dropout, BM and average pooling are used for regularization to speed up the training set. Because CNN shares convolution kernels, there is no pressure on high-dimensional data processing, and it can effectively reduce the dimensionality of pictures with large amounts of data into small amounts of data. In addition, CNN can automatically perform feature extraction. The convolution kernel in the convolution layer can play a role that can extract the required features. After training

the weights, the feature classification could work well. Therefore, CNN can effectively retain image features, which is in line with the principles of image processing.
The main disadvantages of the CNN algorithm:
In CNN, it requires parameter adjustment along with a large sample size, so it has to use GPU for training. Also, when the network layer is too deep, it is easy to make the training result converge to a local minimum rather than a global minimum when using the gradient descent algorithm. The pooling layer could lose a lot of valuable information and ignore the correlation between the part and the whole.

Experiments

3.1 Random Forest

The graph shows the trend of the accuracy and the running time of this random forest classifier based on each controlled variable. The best params calculated by the model shows that when max_depth reaches 110 (the preset maximum value) and n_estimator is 228, the model achieves the best performance.

As the x-axis max_depth increases, before max_depth reaches about 20, the accuracy and running time show a rapid upward trend, accuracy rose to over 74%. Then accuracy increases and running time decreases very slowly by a slight amount, within the value of 75%, accuracy rate slowly increases in the form of a decimal point, and the highest value reaches 75.75%. The runtime dropped slightly but stayed above 330. Those may be hard to see in the curve, but it is true. The intermediate results of model training and the calculation results of best parameters also reflect the slow increase/decrease in accuracy/running time as max_depth changes.

Adjusting max_depth to increase may not have much help in improving the accuracy, or even the improvement is very small, but we still set more max_depth in order to let the model cover more test cases.

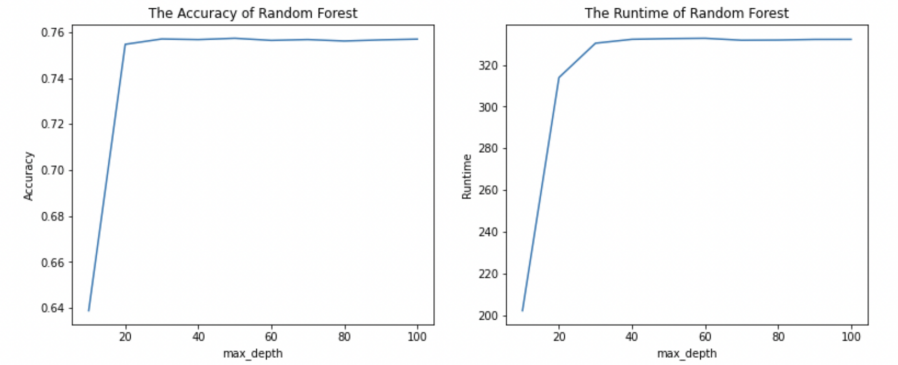


Figure 8 The accuracy vs. max_depth and the runtime vs. max_depth

For n_estimators, as the x-axis increases, the accuracy rate and runtime show a rising or decreasing trend. There is a visible improvement in the accuracy rate(from 73% to 75.5%). Meanwhile, the computation time increases accordingly, a linear growth between n_estimators and runtime are shown below, so the best parameters for the model are able to strike a balance between accuracy and running time. When the value is 221, the maximum score is obtained, accuracy reaching 75.75%.

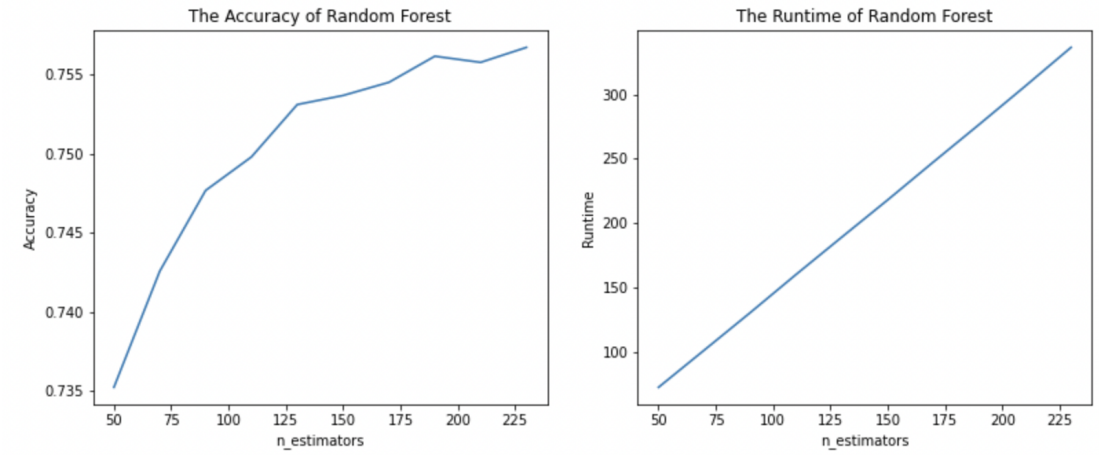


Figure 9 The accuracy vs. n_estimators and the runtime vs. n_estimators

Random forest is a form of random sampling. When calculating each tree, a data set of the same size n that may be repeated is selected for training from all training samples (the number of samples is n), which is bootstrap sampling. A subset of all features is randomly selected at each node and used to calculate the optimal split. Another important part is that after training, Random forests can provide which features are more important, which is helpful for our feature analysis. And random forests can detect the influence between features during training. Overall, Random forests perform well and have advantages over other tree classifiers and simple classifiers.

3.2 SVM

The graph shows the trend of the accuracy and the running time of SVM algorithm based on parameter C. In the comparison diagram of four different kernel functions of SVM as below, the results show that among the four kernel functions of SVM, ‘linear’ has the highest accuracy and the best result, while ‘sigmoid’ has the lowest accuracy with the worst effect. At the very beginning, the accuracy of all these four kernel functions increased rapidly. After the accuracy rate of linear reaches the maximum value for each kernel function, the curve tends to be stable, and the accuracy remains the same no matter how parameter C grows. For another three kernel functions, as the different values of c increases in the figure, the accuracy will get better and better.

It can be clearly seen from the plots below that as the x-axis parameter C goes up, the running time is decreasing, and it has been in a downward trend from the beginning to the very end. According to what is shown in the graph, rbf, poly and sigmoid took about 25,000s to run at the beginning and all of these functions had similar running time while the linear kernel function only used less than 10,000s to run with achieving the highest accuracy score 76.39%.

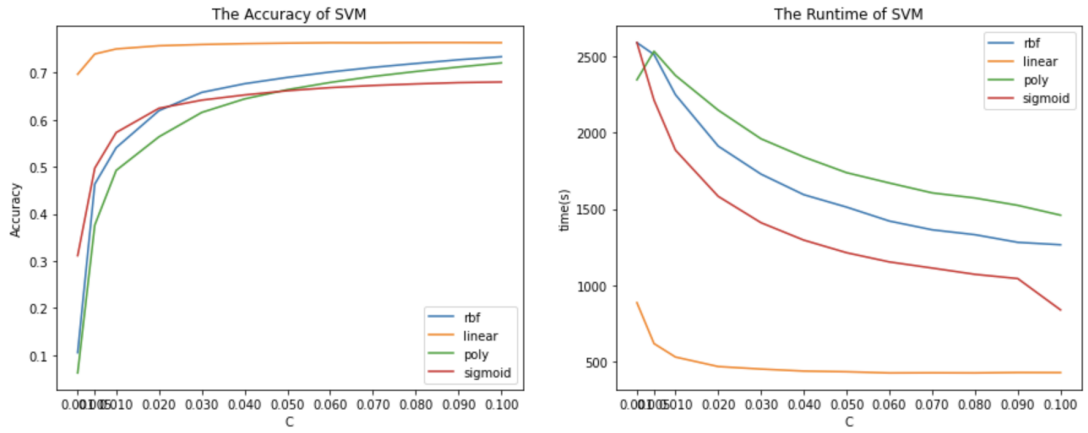


Figure 10 The accuracy vs. c and the runtime vs. c

Compared with other algorithms, SVM is a regression and support vector machine algorithm that can map a dataset into a multidimensional plane by adjusting the settings of kernel function parameters. After transforming a linearly inseparable problem in two dimensions into a linearly separable problem in multiple dimensions, then find an optimal cutting plane, which is equivalent to find an optimal solution based on a decision tree. Consequently, the classification effect of svm may be better than some other simple classifiers.

3.3 CNN

We evaluate the process of the convergence of the model. And through the confusion matrix to analyze the accuracy of the model and analyze some classes where the model training result performs poorly.

Due to the computer configuration environment, we did not perform grid search and fine tone of CNN, because to run the grid search and fine tone of CNN requires high memory capacity， but the environment of our GPUs and laptops limits the speed or in some cases doesn't run at all.

For the parameter analysis of random forest and SVM, each parameter has been converged. For example, like max_depth, each set of data that can be seen is the result of convergence (SVM is the same). And models such as SVM and random forest have a high probability of being able to converge. But neural networks such as CNN are different, it may not necessarily converge. And since it is an iterative optimization algorithm for gradient descent to find the best result (minimum of the curve), we think it is necessary to analyze the accuracy and loss related to epoch and learning rate changes with the training step.

As shown in figure 11, as the epoch increases, the corresponding accuracy also increases. At 0-6 epoch, the accuracy rate is increasing, although there is a fluctuation in the curve, but overall it shows a sharp increase trend. From 6-12 epochs, the rate of increase starts to slow down, and the accuracy keeps rising to more than 87%. Then when the epoch is 12-16, although the accuracy changes slightly, it generally increases, and the rate of increase becomes quite slow. The larger the x-axis, the flatter the trend.

From this, it can be inferred that with the increase of epoch, the model gradually moves towards the trend of convergence. At 12 epochs, the model converged.

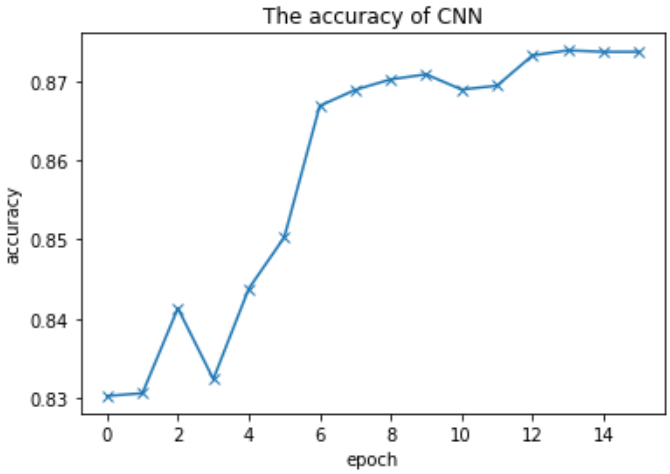


Figure 11 The accuracy vs. epoch

Figure 12 is loss vs. epoch. It can be clearly seen that with the increase of epoch, the loss shows a continuous downward trend. 'loss' is the loss value calculated by our pre-set loss function. 'Loss' is expected to be as low as possible. It can also be reflected from the image that our model is basically successful.

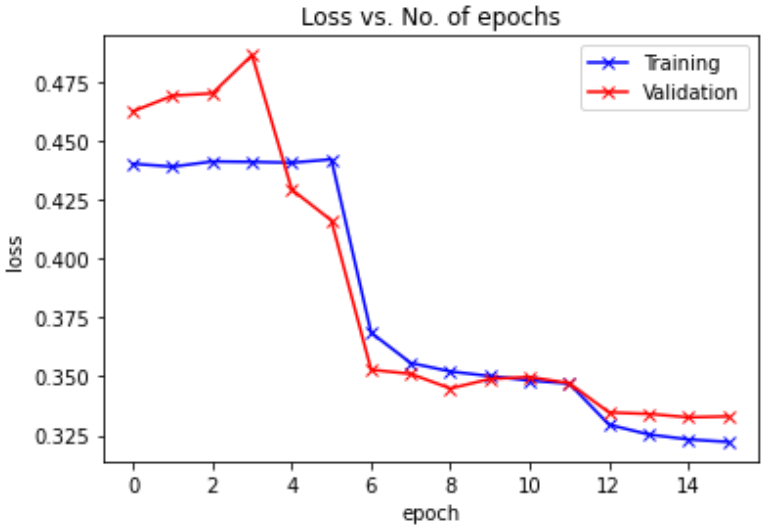


Figure 12 The loss vs. epoch

Figure 13 below shows the change curve of the learning rate with the training step. We use StepLR as the scheduler, and the learning rate decay coefficient is 0.1.

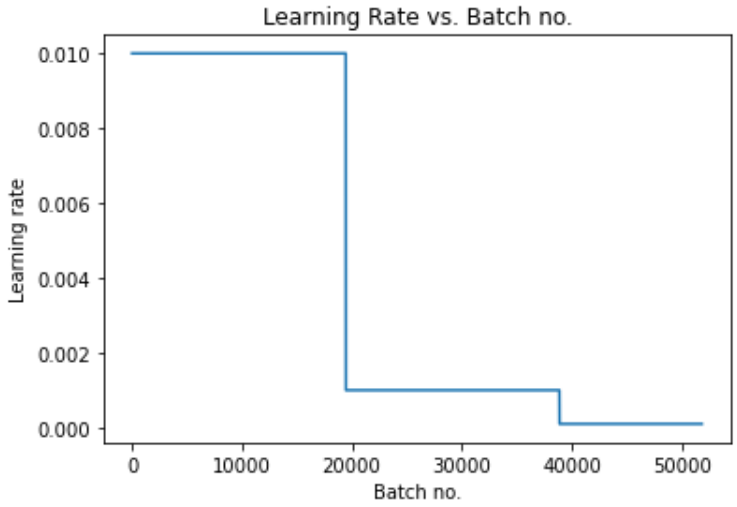


Figure 13 learning Rate vs. Batch no.

The following is to check the accuracy of the model and analyze the prediction results through the confusion matrix. The first matrix is the accuracy of each class and the ranking of the accuracy of all class from small to large, and the second matrix corresponds to the index of each class of the above matrix.

It can be seen that the class with particularly poor accuracy is 50 38 48, etc.

```
[0.0, 0.004739336492890996, 0.010810810810810811, 0.015706806282722512, 0.05405405405405406, 0.085, 0.171226831421006
17, 0.3333333333333333, 0.3620689655172414, 0.42934782608695654, 0.45161290322580644, 0.49491525423728816, 0.51648351
64835165, 0.5444997236042012, 0.5808383233532934, 0.6020942408376964, 0.6111111111111112, 0.6616161616161617, 0.66964
28571428571, 0.6884422110552764, 0.7282608695652174, 0.7314814814814815, 0.7515723270440252, 0.7946428571428571, 0.80
12048192771084, 0.8229376257545271, 0.8481375358166189, 0.8961424332344213, 0.9025157232704403, 0.9032258064516129,
0.910394265232975, 0.9229661627069834, 0.9380403458213257, 0.9388646288209607, 0.9400921658986175, 0.940577249575551
8, 0.9419924337957125, 0.9421422986708365, 0.9425287356321839, 0.9451303155006858, 0.953198127925117, 0.9633173843700
159, 0.9636871508379888, 0.9651741293532339, 0.9653739612188366, 0.9670212765957447, 0.9761092150170648, 0.9774220032
840722, 0.9777777777777777, 0.9778850020104544, 0.9779735682819384, 0.9790419161676647, 0.98014440433213, 0.982783357
2453372, 0.9838619922092376, 0.984, 0.9845360824742269, 0.985495234148363, 0.986, 0.9918256130790191, 0.9945076500588
466, 0.9956913435174305]
[50 38 48 54 56 41 47 57 51 44 52 18 60 24 46 61 42 35 45 58 33 59 31 34
 20  0 32 37 13 16 19  1 29 36  5 25 49 55 21 28 15 43 53 26 12 30 11  6
 10  2 22 27 23 39 40 17  4  9  8 14  7  3]
```

Figure 14 Confusion matrix of CNN

The following is the analysis of the reasons for the poor precision of these classes.

The 50th class corresponds to a lowercase 'o'. The vectors shown in the figure below represent the model recognition results for the lowercase ‘o’ samples in the test set.

The 0th class and the 24th class, corresponding to 86 data and 111 data, are the two classes with the most model recognition results. They correspond to the number 0 and the uppercase 'O' respectively. In the recognition result of lowercase 'o', there are some numbers ‘0’ data and uppercase 'O' data appear. In other words, the model incorrectly recognizes these two types of data as lowercase 'o'. And these mistakes are the reason for the accuracy of the model not reaching 100%.

```
# analysis 50
print(dataset.classes[50])
# confusion matrix for 'o'
print(matrix[50])
print('86 wrong result :' + dataset.classes[0])
print('111 wrong result :' + dataset.classes[24])
```

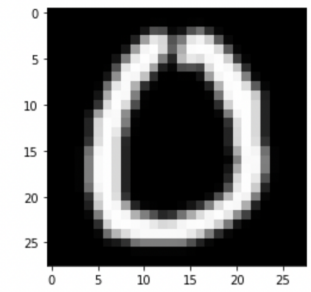
```
o
[ 86   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0
   1   0   0   0   0   0 111   0   0   0   0   0   0   0   0   0   0   0
   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0]
86 wrong result :0
111 wrong result :O
```

Figure 15(a) Analysis for training result of CNN

As can be seen from the original data image, the image on the left is a lowercase 'o', the image number 0 in the middle, and an uppercase 'O' image on the right. They really look alike, and even the human eye seems to be difficult to tell them apart, so it's understandable that the model misidentifies them.

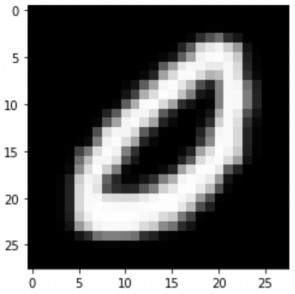
```
count = 0
for (img, label) in test_dataset:
    if label == 50:
        if count == 0:
            show_example((img,label))
            break
        count += 1
```

Label: (o)



```
count = 0
for (img, label) in test_dataset:
    if label == 0:
        if count == 0:
            show_example((img,label))
            break
        count += 1
```

Label: (0)



```
count = 0
for (img, label) in test_dataset:
    if label == 24:
        if count == 0:
            show_example((img,label))
            break
        count += 1
```

Label: (O)

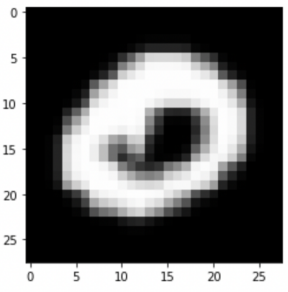


Figure 16(a) Original image of CNN

This is the precision result of lowercase 'c' in the test set. It can be seen that 203 data are recognized as uppercase 'C'.

```
# analysis 38
print(dataset.classes[38])
# confusion matrix for 'c'
print(matrix[38])
print('203 wrong result :' + dataset.classes[12])
```

```
c
[  0   0   0   0   0   0   0   0   0   0   0   0 203   0   0   0   0   0
   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   1   0   1   0   2   0   0   0   1   0   0   0   0   0   0   0   0   2
   0   0   0   0   0   0   0   0]
203 wrong result :C
```

Figure 15(b) Analysis for training result of CNN

The image on the left is a lowercase 'c', and the image on the right is an uppercase 'C'. They are really similar.

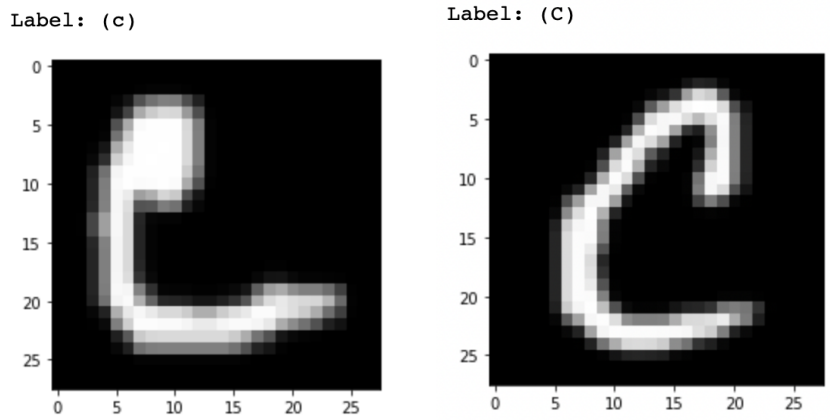


Figure 16(b) Original image of CNN

The precision results of lowercase 'm' in the test set showed that 181 data were incorrectly recognized as uppercase 'M'.

```
# analysis 48
print(dataset.classes[48])
# confusion matrix for 'm'
print(matrix[48])
print('181 wrong result :' + dataset.classes[22])
```

```
m
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0 181   0   0   0   0   0   0   0   0   0   1   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   2   1   0   0   0   0
   0   0   0   0   0   0   0   0]
181 wrong result :M
```

Figure 15(c) Analysis for training result of CNN

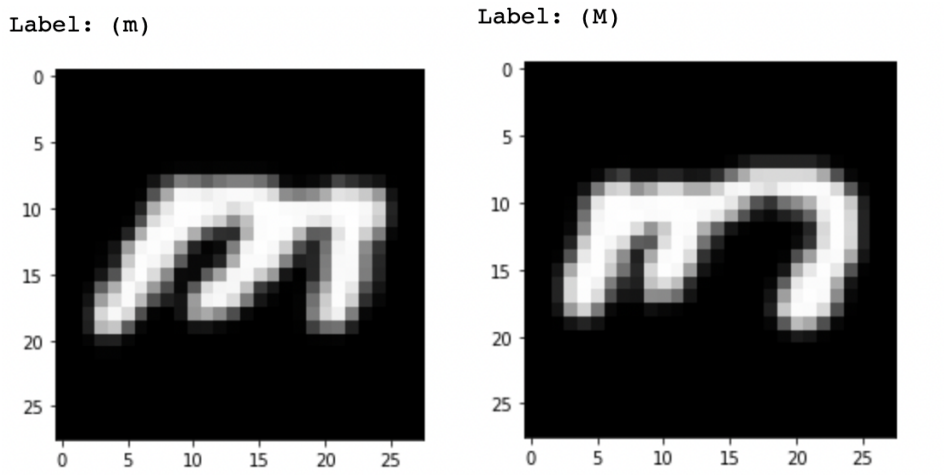


Figure 16(c) Original image of CNN

After continuous research, we believe that there is no better optimization method for the above identification errors from the perspective of the algorithm, but can only be solved from the data itself, for example, by artificially adding case-sensitive labels when inputting data.

Performance Metrics of 3 models

Common evaluation indicators for machine learning classification models are Accuracy, Precision, and Recall. In binary and multi-class models, the calculation methods of evaluation indicators are slightly different.

	SVM	**RF**	**CNN**
Accuracy	75.75%	74.79%	87.36%
Recall	57.33%	48.25%	75.26%
Precision	62.04%	64.65%	81.08%

As shown in the table, Precision and Recall are often a pair of contradictory performance metrics. Increased precision means improving classifier predictions to ensure what the classifier predicts is as real as possible .Increasing the Recall means lowering the classifier prediction, so that the classifier can pick out the real positive examples as much as possible.

Conclusion

In order to ensure that each algorithm achieves the optimal solution and finally compare and pick the best model among these different classifiers, this experiment includes loading training data and testing data, pre-processing data, implementing algorithms, building classifiers, tuning hyper-parameters, calculating accuracy score and running time, and predicting results. After using cross-validation and random validation in different algorithms, then adjusting the parameters to find out the highest accuracy score with a reasonable running time.

Since classifier learning requires a large amount of training data and a learning algorithm with strong generalization ability, the size of the training data is critical when choosing a classifier. According to the complexity of the problem and the size of the training data, a good classifier should be able to automatically adjust the balance between the ability of fitting and the generalization. Furthermore, the complexity of the classification function and the size of the training data and the dimension of the input feature space need to be considered. For the perspective of the future, when pursuing high accuracy, it is better to use a more complex model instead of a basic classifier, such as Convolutional Neural Network.

One of the basic classifiers, Random Forest, a new flexible and practical method that has a wide range of application prospects could run large databases efficiently with high accuracy. The greater the entropy, the greater the uncertainty of the category, and vice versa. The random forest integrates all the classification voting results and assigns the category with the most votes as the final output, which is the simplest Bagging idea. Even if a large proportion of the data are missing, Random Forest generates an internal unbiased estimate of the generalization error, which could estimate missing data and maintain accuracy due to the strong ability of generalization. As a result, Random Forest outperforms any single-class prediction. It is fast and parallelized to train data using Random Forest because it can process high-dimensional data and perform different feature selection and handle unbalanced data and balance errors. However, when the data noise is relatively large, overfitting will occur.

Another example is Support Vector Machines. The usual step in classification using SVM is to first manually extract features such as size, shape, weight, color, etc. According to the above features, each image is mapped to a point in the space, and the dimension of the space is equal to the number of features. Because objects of the same class have similar characteristics, points in space labeled as one class must be clustered together. At this time, a basic classifier such as the SVM algorithm is used to draw a dividing line between various points in the space to complete the classification. In the case of a large picture size, the calculation amount will be large and difficult to estimate. Not all the information in the picture is needed, for example the background is worthless for classification. However, in the base classifier, it will all be used as input. The background is also entered into the model as a feature, and the accuracy will naturally decrease. In general, if the features are not extracted manually, the amount of calculation will be very large, and the accuracy cannot be guaranteed.

CNN has successfully solved all problems in Random Forest and SVM in a unique way. A neural network can automatically extract features and has relatively few parameters to be trained. Using convolution to simulate how the human visual system works, which is the decisive advantage of CNN in image classification tasks. This method greatly reduces the number of parameters to be trained in the neural network. To obtain translation invariance, a weight sharing technique would be used, which further reduces the number of parameters to be trained. The convolutional neural network uses the original image as input, which can effectively learn the corresponding features from a large number of samples and avoid the complex feature extraction process. Because the convolutional neural networks can directly process two-dimensional images, it has been widely used in image processing and has achieved many research results.

References

1. Belgiu, M., & Drăguţ, L.Random Forest in remote sensing: A review of applications and future directions. ISPRS journal of photogrammetry and remote sensing 2016; 114: 24-31.

2. BREIMAN, L. Random forests—random features. Technical Report 567, Statistics Department, University of California, Berkeley 1999; 2:123-140.

3. Chen Yan, Chen Yalin, & Zheng Jun. An Improved PCA-Based Data Preprocessing Method. Electronic Technology Applications 2020; 4:1-2.

4. Classification Evaluation Indicators: Accuracy, Precision, Recall, F-measure.
Towards Data Science 2020; 12:2-3.Yukkuri Machine Learning 2022; 61:2-3.

5. Cohen G, Afshar S, Tapson J, van Schaik A. EMNIST: an extension of MNIST to handwritten letters. Cornell University 2017;5:1-2.

6. Jayaswal V. Performance Metrics: Confusion matrix, Precision, Recall, and F1 Score.Towards Data Science 2020; 12:2-3.

7. Loukas S. Everything you need to know about Min-Max normalization: A Python tutorial.Towards Data Science 2020; 16:1-2.

8. PAL, M. Random Forest classifier for remote sensing classification. Informa UK Limited 2022; 6:1-2.

9. PAL, M. and MATHER, P.M. An assessment of the effectiveness of decision tree methods for land cover classification. Remote Sensing of Environment 2003a; 86: 554–565.

10. Palaniappan, R., & Ravi, K. V. R. Improving visual evoked potential feature classification for person recognition using PCA and normalization. Pattern Recognition Letters 2006; 27:726-733.

11. Sharma, Sagar. Epoch vs Batch Size vs Iterations.Towards Data Science 2017; 43:1-2.

12. Wright, Matthias. PyTorch ResNet9 for CIFAR-10.GitHub, Inc.2022; 1-2.

Appendix

Computer Configuration:

Device name: LAPTOP-6LDCFT6G
Processor: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
On-board RAM: 8.00 GB (5.92 GB available)
Device ID: 2514707B-0189-482C-BCAD-654D689C6B33
Product ID: 00342-35848-84625-AAOEM
System Type: 64-bit operating system, x64-based processor
System Manufacturer: HP
System Model: HP 245 G7 Notebook PC
BIOS: F.54 (type: UEFI)
Pen and Touch: No pen or touch input available for this monitor
Memory: 8192MB RAM
Available OS Memory: 6058MB RAM
Page File: 5660MB used, 2303MB available
Card name: AMD Radeon(TM) Vega 8 Graphics
Manufacturer: Advanced Micro Devices, Inc.
Chip type: AMD Radeon Graphics Processor (0x15D8)
DAC type: Internal DAC(400MHz)
Device Type: Full Device (POST)

How to Run Code:

1. Run Load Data Section
2. Run Preprocessing Normalization Section
3. Run Preprocessing PCA Section
4. Run Random Forest Classifier
5. Run Support Vector Machines Classifier
6.Run Convolutional Neural Network Classifier with Graphics Processing Unit support