Title
the image classification of MNIST dataset by using machine learning techniques

Permalink
https://escholarship.org/uc/item/0505d532

Author
Liu, Yang

Publication Date
2021-12-15

UNIVERSITY OF CALIFORNIA,

MERCED

The image classification of MNIST dataset by using machine learning techniques

REPORT

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical Engineering and Computer Science

by

Yang Liu

Master Report Committee:

Professor Alberto Cerpa
Professor Wan Du
Professor Shawn NewSam

2021

DEDICATION

I dedicate all my work to Professor Alberto Cerpa, who helped
me from the beginning to the end with great patience and care.
Besides I want to express my appreciation for all who helped me in finishing
the project, like Devanshu Kumar,Professor Wan Du, and Professor Shawn NewSam.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ABSTRACT OF THE REPORT

The image classification of MNIST dataset by using machine learning techniques

By

Yang Liu

MASTER OF SCIENCE in Electrical Engineering and Computer Science

University of California, Merced, 2021

Professor Alberto Cerpa, Chair

In this report several classifiers are applied to predict the class of MNIST testing dataset and compare their performance. First, training dataset is split into training and validation sets so that cross validation can be applied to determine the best hyperparameters for the models used and the best size for training. Second, by using the best hyperparameters, the models are trained on the dataset with appropriate size. Finally, the trained models are applied on testing dataset to compare the models' performance on unseen data.

Chapter 1

Introduction

The MNIST dataset is well known public dataset which consist of handwritten digits. It has a training set which has 60,000 examples and a test set of 10,000 examples. [4] It was created by remixing the samples taken from American Census Bureau employees and American high school students. The training set contains 30,000 images from employees and 30,000 images from students while test set contains 5,000 images from employees and 5,000 from students.[5]

After it was created, several methods have been tested with training set and test set while the test errors were recorded. Here are test errors of some kinds of classifiers [4]

There have been so many previous works on dealing with mnist datasets. In 2018, Dr.Young Im Cho From Gachon University published a paper concerning the performance

CLASSIFIERS	LOWEST TEST ERROR RATE	HIGHEST TEST ERROR RATE
LINEAR	7.6	12.0
K-NEAREST	0.63	5.0
BOOSTED STUMPS	0.87	7.7
NON-LINEAR	3.3	3.6
SVM	0.56	1.4
NEURAL NETS	0.35	4.7
CONVOLUTIONAL NETS	0.23	1.7

Table 1.1: the lowest and highest test error of different classifiers

of Regression Model, CNN, ResNet and their own model CapsNet. They manage to achieve a 99.4% accuracy for their own CapsNet model, which are higher than other models.[6]

Most of the work related to mnist involves the neural network and its improvement. I rarely find someone using traditional classfiers on it. So in this report I will try to use some traditional Methods on mnist datasets and compare their performance with cnn.

This report will show the following work

1. Finding the best hyperparameters

Beside the linear classifiers most techniques have parameters that requires fine-tuning. By using part of the data for training model and other data for validation, the best combination of hyperparameters will be chosen based on the validation error.

2. Find the models that has the best performance

By applying the models trained from training set on the test data, the performance of models will be ranked based on their prediction on test data.

The remainder of the report will be organized as following. In chapter 2, the background of techniques will be discussed. In chapter 3, the experimental setups and details of work will be described. In chapter 4, the performance of models will be shown. In chapter 5, the results of experiment and the performance of models will be discussed.

Chapter 2

Background

2.1 Classifiers

2.1.1 Bayesian Decision

Bayesian decision theory is an approach based on quantifying the tradeoffs between various classification decisions using probability and the costs that accompany such decisions.[**?**] Below is an example of how Bayesian decision works

Suppose we are trying to determine the type of incoming fish on a belt. We have 2 kinds of fish, salmon and carp. We denote n the nature of the fish, with $n = n_1$ for salmon and $n = n_2$ for carp.

If we have caught enough carp and salmon, we can determine a prior probability $P(n)$. If we have caught 6,000 salmon and 4,000 carp, the probability $P(n_1) = 6,000/(6,000 + 4,000) = 0.6$, and $P(n_2) = 0.4$

Then we can introduce a variable called lightness, which is denoted by x. It is a continuous value. The distribution of lightness under fish type can be expressed By a class-conditional probability density function: $P(x|n)$. $P(x|n1)$ is the probability density function of salmon fish.

If we have prior probability $P(n)$ and probability densities $P(x|n)$, how to predict the type when we get a lightness x.

By using Bayes' formula:

$$P(n) * P(x|n) = P(x) * P(n|x) \tag{1}$$

we have $P(n|x) = P(n) * P(x|n)/P(x)$

So In this case we can get a posterior probability $P(n|x)$ from prior probability $p(x|n)$

For real problems, the feature x will be a n-dimension vector, which depends on how many features are available.

In my program I use GaussianNB, which supposes that the input features follow gaussian distribution. It will try to calculate a probability distribution function for each feature and calculate the posterior probability from that.

2.1.2 K-Nearest-Neighbor Rule

This rule classifies target x by assigning it the label most frequently represented among the k nearest samples.

The k-nearest neighbor query starts at the test point and grows a spherical region until it encloses k training samples. The number k is the hyperparameter in this algorithm. We can select the appropriate k for algorithm based on validation.[**?**]

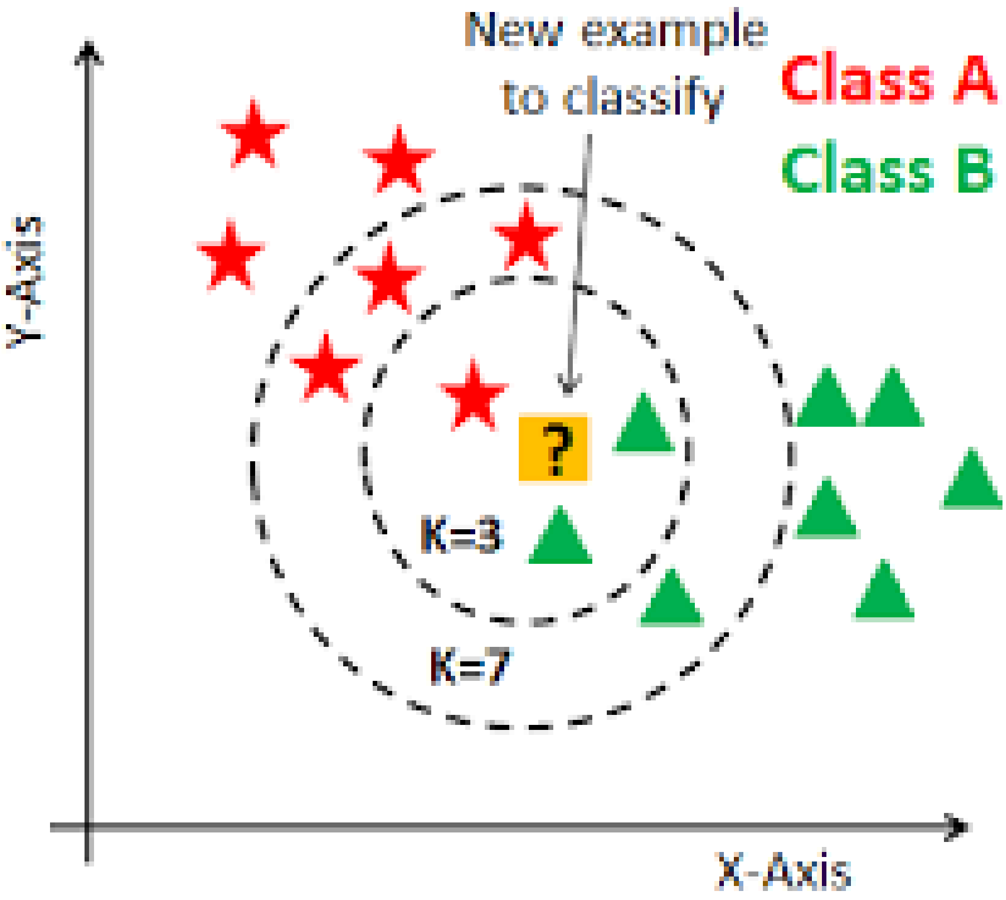


Figure 2.1: An example of how k-nearest neighbor algorithm works [1]

In my program I directly use the sklearn's KNeighborsClassifier() API. I have applied PCA decomposition on the input, so every single point of an input will be a 154-d point. (the ideal component for my PCA reduction is 154).

The KNeighborsClassifier() is a non-parametric method, which means it does not train any parameters in the process of fitting training data. Instead, it just remembers all the training data into its model. When a new test data point is coming, it will first calculate the Euclidean distance of the new point and every single training point. If you have 60,000 points of training data, then you need to compute 60,000 distances for a single test point. The distances will then be ranked in order to get the k-smallest distances. The training points that correspond to those smallest distances will be taken out.

Imagine you set k=5, now you have 5 points. Their class labels are 1,1,0,1,0. Then the test data's label will be assigned 1, which is the most common among the 5 points.

The label of the test point will be assigned the most frequent label in its nearest neighbors. As its distance is calculated by using Euclidean distance, the data on different columns needed to be rescaled into the same range.

2.1.3 Support Vector Machine

Support Vector Machine is a linear classifier.

A labelled data is called linearly separable if there exists a linear decision boundary separating the classes.

There can be many decision boundaries that separate the classes, but some are better than others.

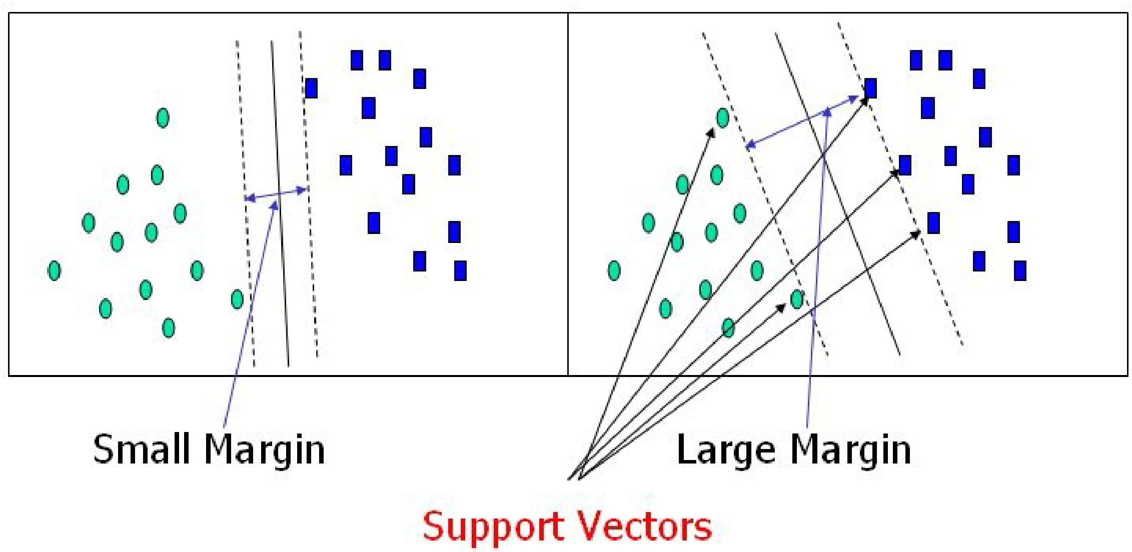


Figure 2.2: The different decision boundaries

In this problem, we define a concept called margin, which is the perpendicular distance of the point to the decision boundary. In this case we denote margin M.

The target of Support vector machine is to find the decision boundary that maximizes the value of margin.

Support vector machine is used for binary classification. For multi classification, it first breaks down the problem into several binary classification problems. [6]

the vector that lies on the border are called support vectors.

Imagine we have a vector w, and $y = w * x + b$ will be the decision boundary.

And from calculation, it can be concluded that M = $1/||w||$.

So maximizing the margin means minimizing wTw. Also all the points needs to be divided. Which requires $y_i(w^T x_i + b) >= 1$

in which y_i is the label for the point x_i

In the real scenario there may not exist a line or hyperplane that divides All the data. In this case, we change the problem into a soft-margin classifier.

Which will fit $y_i(w^T x_i + b) >= 1 - e_i$, in which e_i is the error tolerated for point i. For the function we try to minimize, it becomes $L(w, e) = w^T w + C \sum_{i=1}^{N}(e_i)$

In which C is the penalty coefficient for error tolerance. The bigger the C, the tolerance for error becomes lower, but has the possibility of overfitting. The smaller the C, the model will have higher tolerance for errors, which Mean lower performance but better generalization.

C is a hyperparameter in SVC.

Another important aspect is kernel. Sometimes by transforming data we may be able to make the data from unseparable to separable.

For example we can use q(x) instead of x as the input features. The function q(x) is the kernel function.

We can have polynomials or linear function as our kernel function. In SVC, The gaussian kernel is used.

Another hyperparameter used in my model is the gamma, which is the parameter For the kernel function.

I use SVC as my model, and there are two parameters that requires tuning: C and gamma. I use Gaussian kernel as my kernel function.

2.1.4 Decision Tree

Decision Tree is based on the binary tree, and its cost is relatively low with $O(\lg n)$, in which n is the number of datapoints.

Decision tree works by breaking classification down into a set of choices about each feature in turn, starting at the root of the tree and going down to leaves, in which each leave is a classification decision.

The commonly used algorithms are ID3 and CART (Classification and regression trees).

ID3 uses information gain as its metric.

In information theory, the entropy of a probability p is Defined as the following

$$Entropy(p) = -\sum_{i=1}^{k} p_i * \log_2 p_i \tag{2}$$

in which k is the different possible value of the target and p_i is the probability of i_{th} possible value

ID3 tries to work out how much the entropy of the Whole training set will decrease by choosing particular feature for classification, which is known as the information gain. The larger than gain, the better the feature.

ID3 computes the information gain for each feature and chooses the one that produces the highest value.

The CART uses Gini Impurity as the metric.

Gini impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable. Its lower bond is 0, meaning that all datapoints belong to one class.

The Gini impurity for feature k is defined as the following:

$$G(k) = \sum_{i=1}^{c} \sum_{j \neq i} N(i) * N(j) \tag{3}$$

In which c is the number of classes, and $N(i)$ is the fraction of the datapoints that belongs to class i. The CART will try to reduce the $G(k)$ in constructing the decision tree.

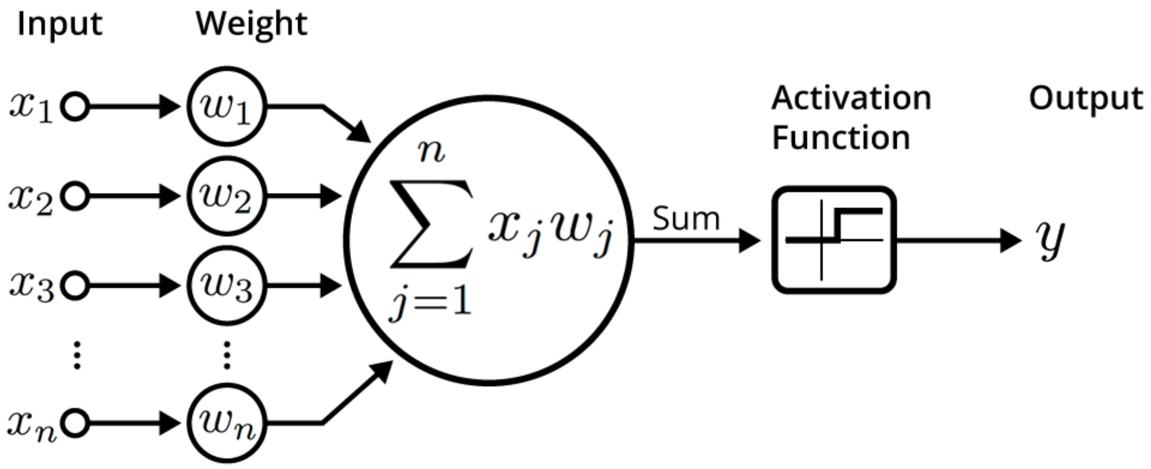
2.1.5 Random Forest

The random forest is an ensembled model of multiple decision trees. It trains its own model by doing the following: For each tree: 1. Create a subset of the training set 2. Use the subset to train a decision tree 3. for each decision tree, uses part of the feature rather than all the features to split the node 4. repeat until the tree is complete

Its output will be the most frequently output of all trees in the model

2.1.6 Convolutional Neural Network

A neuron, which originates from the neuron cell from human brain, is a small processor which can accept input signals and fire an output. The neurons can be connected to each other in a synapse.



An illustration of an artificial neuron. Source: Becoming Human.

Figure 2.3: The structure of a neuron

The structure of a neuron in machine learning consists the following part

(1) The inputs xi

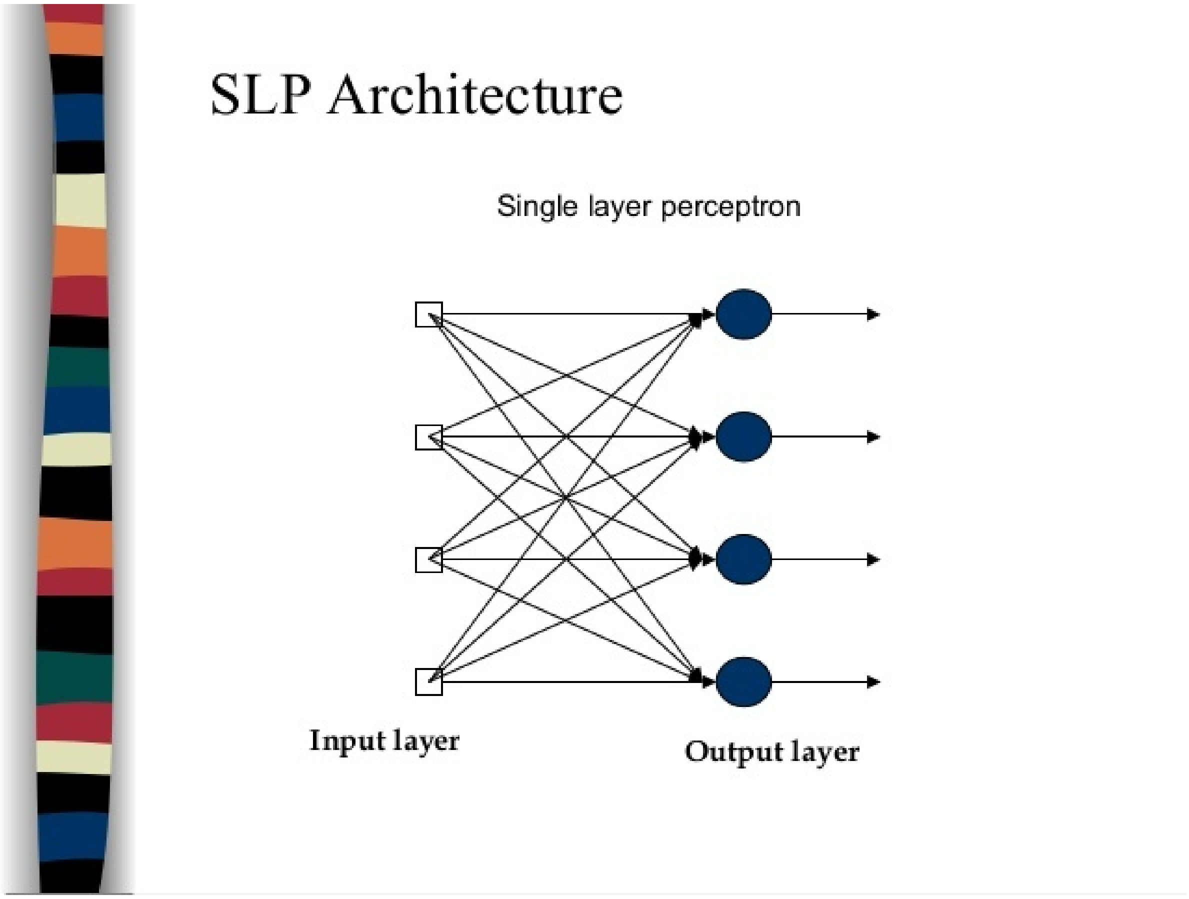
(2) A set of weighted inputs wi

(3) An adder that sums the weighted input signals

(4) An activation function that decides whether the neuron fires for the current inputs

The activation function will deal with the sum of the inputs. Some common activation activation functions include sigmoid, tanh, ReLU.

Usually a single neuron can do so much in the machine learning. In order to strengthen its power, neurons are always combined in a network.



SLP Architecture

Figure 2.4: The structure of a single-layer perceptron

The neurons in perceptron accepts the same input from input layer. The difference Between neurons is that different neurons will have different weights or different activation function. The neurons are independent on each other, only having the same inputs.

Usually a perceptron with only 1 layer does not have really strong learning capability, in order to make it stronger, adding more layers between the input layer and output layer, which are called hidden layers, is a good choice.

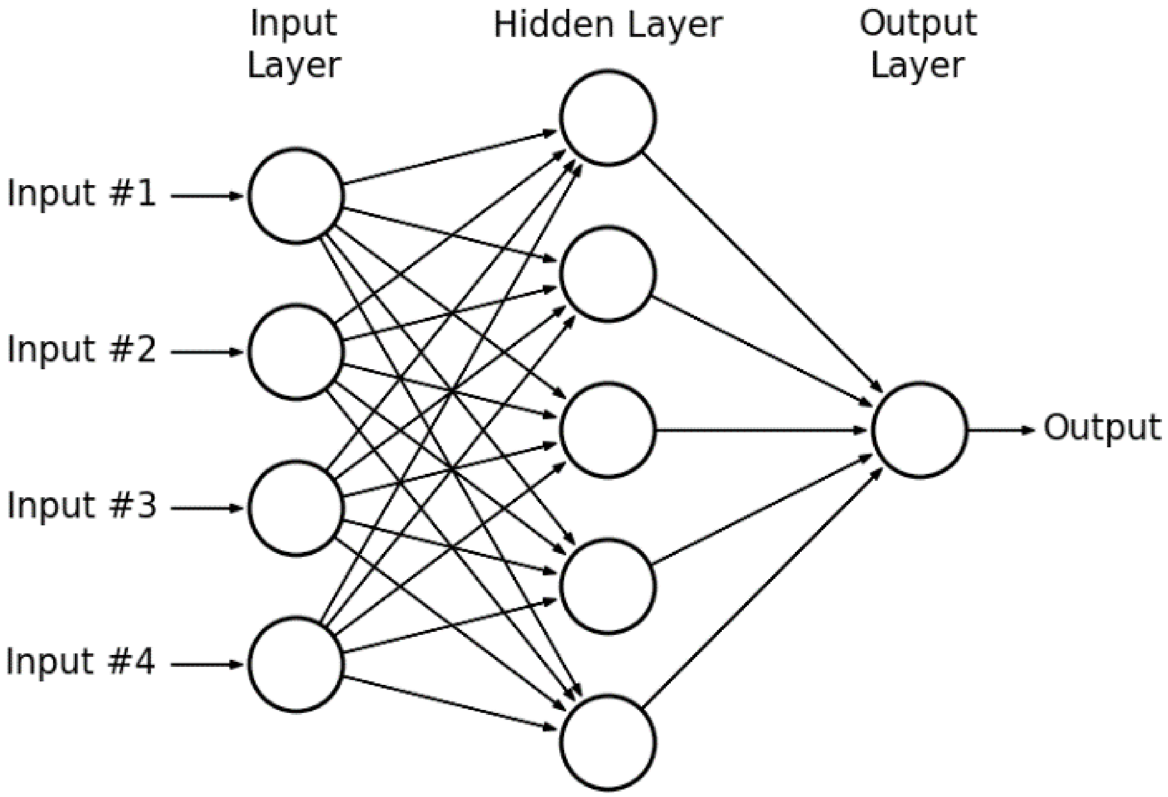


Figure 2.5: The structure of a multilayer perceptron

Multilayer perceptron uses backward propagation to train its model.

Convolutional neuron network, in short called CNN, is a regularized version of multilayer perceptron. CNN uses convolution in place of general matrix multiplication in at least one of their layers. CNN is always used for image classification.

In a CNN, the input is a tensor with a shape: (number of inputs) * (input height) * (input width) * (input channels). After passing through a convolution layer, the output will be a new tensor with shape: (number of inputs) * (feature map height) * (feature map width) * (feature map channels)

Besides convolutional layer there can be pooling layers, which can Reduce the dimensions of data by combing the outputs at one layer Into a single neuron in the next layer. There are two types of pooling which are commonly used. Max pooling(uses the maximum value of the outputs in the feature map) and average pooling(uses the average of the outputs in the feature map).

Convolutional layers and Pooling layers are for feature extraction. After feature extraction The fully connected layer is required for classification, just like the multilayer perceptron. It will connect the information extracted from previous steps to the output layer and eventually classifies the input into desired label.

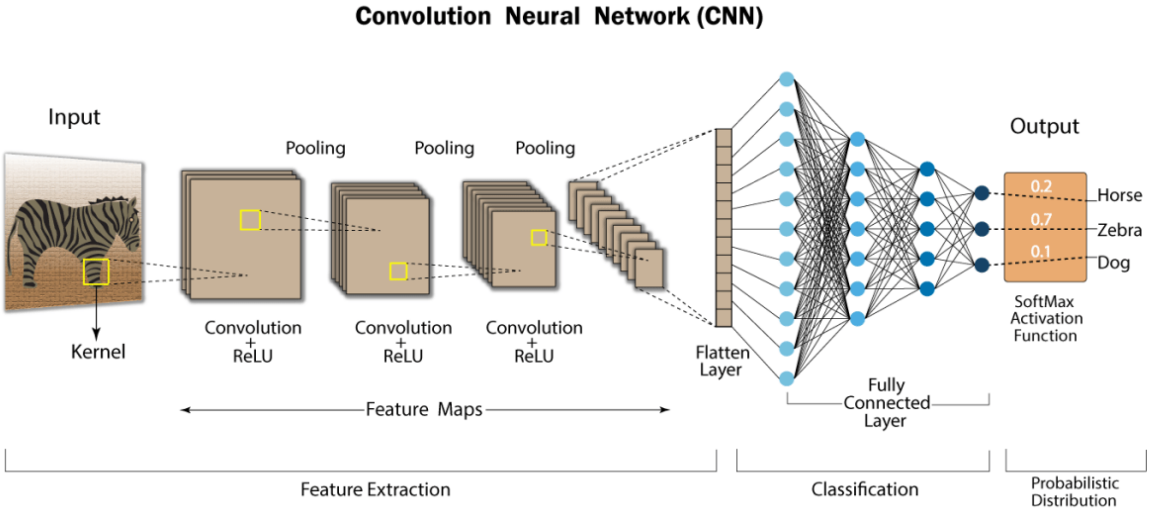


Figure 2.6: The complete process of a CNN

2.2 Metrics

2.2.1 Precision, Recall, F1 score and ROC Curve

The output of classifier can be arranged in a simple chart

True positive	False positive
False Negative	True Negative

Table 2.1: the possible outcome of binary prediction

$$Precision = TP/(TP + FP) \tag{4}$$

$$Recall = TP/(TP + FN) \tag{5}$$

Precision is the ratio of correct positive examples to the number of actual positive examples, while recall is the ratio of the number of correct positive examples out of those that were classified as positive, which is the same as sensitivity.

F1 score is defined as the following F1 = 2(precision*recall)/(precision+recall)

The ROC curve is a plot of true positive rates on the y axis against the false positive rate on the x axis.

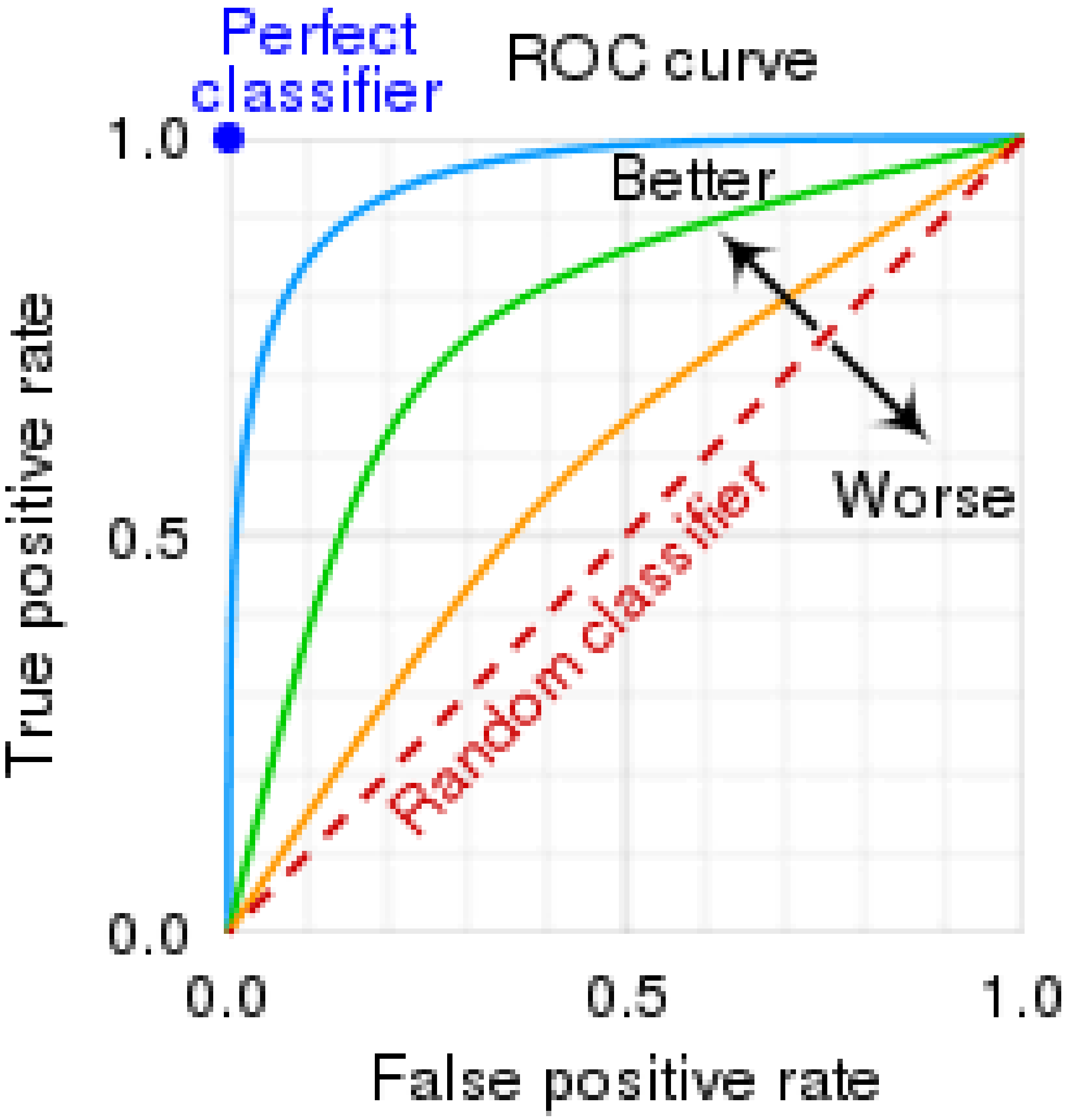


Figure 2.7: Examples of ROC curve [2]

The best classifier will be a point at (0, 1). Usually the Area Under Curve, in short called AUC, is used as a metric for classification problems.

2.3 Cross validation

2.3.1 Validation Set

We use test set to evaluate our model, but test set cannot be touched until we need final evaluation of our model, so what can be done to improve our model? Usually we divide the training set we have into training set and validation set. Like in Fig 2.4.

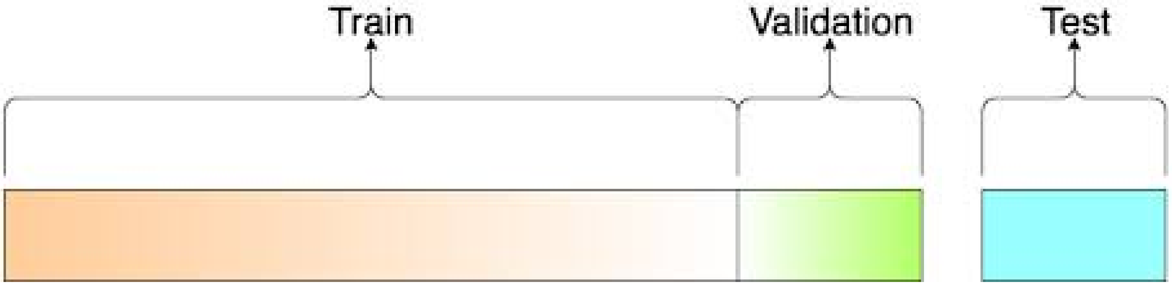


Figure 2.8: Training, validation and test set

Training set is the data we use for training our model, and validation set is the data used for determining if our model needs improvement.

Validation set has the meaning of preventing overfitting

Overfitting is common in machine learning/deep learning, which means having a very good performance on training set but having a poor performance on real data. We can use the model's performance on validation set to determine if its overfitting. If a model continues to have its training error decreasing but validation error increasing, it means that the model is overfitting. The model which has the lowest error on validation set will be selected.

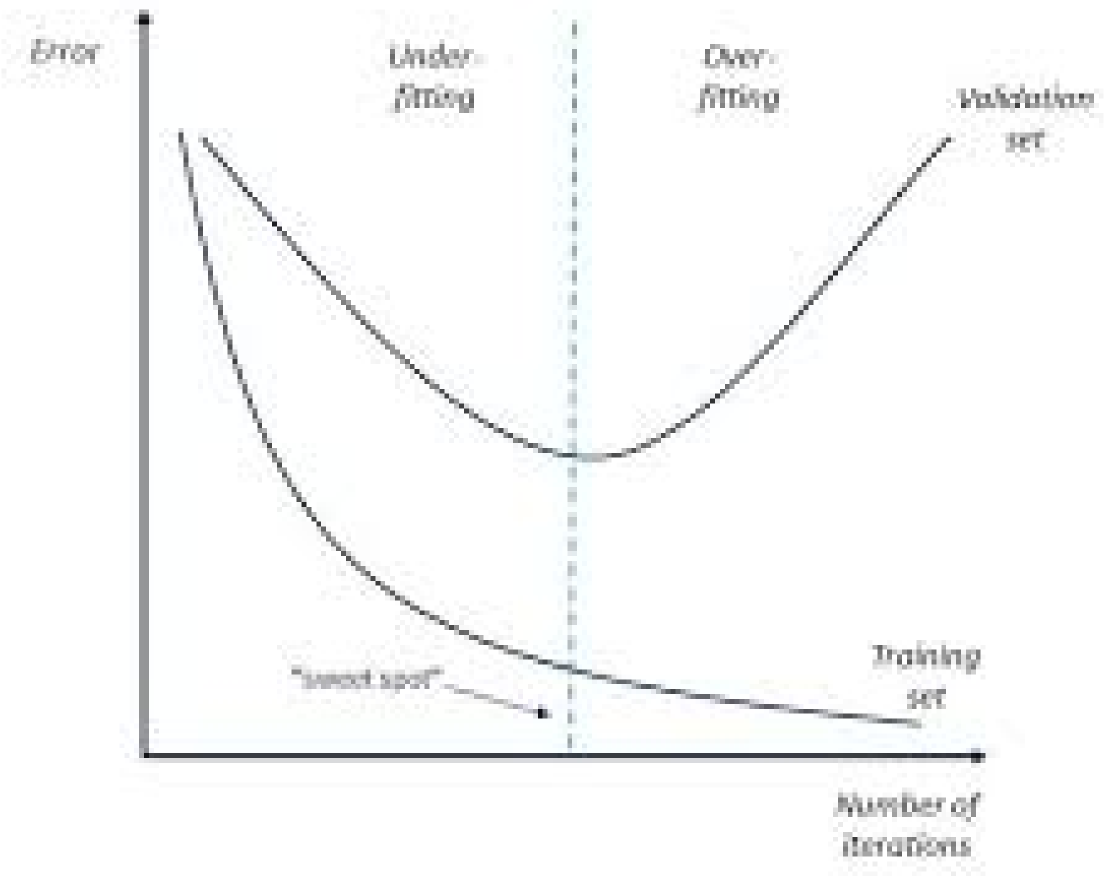


Figure 2.9: An example of overfitting

2.3.2 Validation On Different Datasets

Besides avoiding overfitting, The validation set can be used for tuning the hyperparameters, which cannot be trained in the training set.

In search for the hyperparameters that works best for the model, the training set needs to be split into training and validation set so that models will be trained on training set and their performance on validation set will be used to select the best hyperparameters.

As we have seen in section 2.3.1, the original dataset is just divided into the training and validation set. How we divide the original dataset can have an impact on our outcome.

Imagine if we have 2 types of images, clothes and shoes to detect. We split the data into a training data which are all shoes and validation set which are all clothes, how could the outcome be? No matter how well we behave on training data, we are bound to have poor performance on validation set. How could this be avoided?

Cross validation is a method to solve this. Rather than dividing the original once into training and validation set, we divide the original data several times. Below is an example of cross validation, which is called k-Fold cross validation.

A commonly used cross validation method is the K-Fold cross validation, which divides the training set into k parts with the same size, and train the models k times by taking k-1 parts as the training set and the remaining 1 part as the validation set. The average score of models on k validation sets will be used for selecting the hyperparameters.

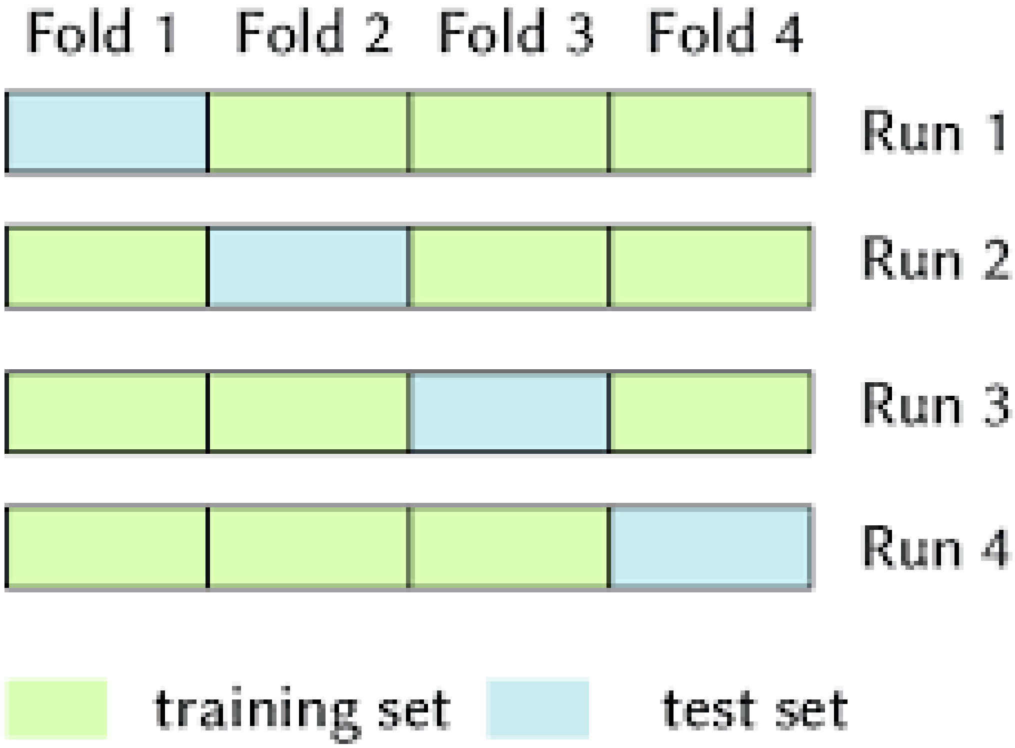


Figure 2.10: An example of 4-fold cross validation

For example, if we have 60,000 examples, and we need to perform 4-fold cross validation, what we need to do is the following:

(1)divide the original data into 4 equals folds which have 15,000 examples in which the proportion of classes remains the same as the original data(eg: if you have 2 classes of data which are 50% :50% in the original data, after division each fold also needs to keep 50%:50% proportion)

(2)Take fold 1 out, use fold 2,3,4 for training models, use fold 1 to validate the Model trained by fold 2,3,4, get a score/loss/

(3)Take fold 2 out, use fold 1,3,4 for training models(use a new unfitted model), use fold 2 to validate the model trained by fold 2,3,4, get a score/loss/

(4) Take fold 3 out, use fold 1,2,4 for training models(use a new unfitted model), use fold 3 to validate the model trained by fold 1,2,4, get a score/loss/

(5)Take fold 4 out, use fold 1,2,3 for training models(use a new unfitted model),use fold 4 to validate the model trained by fold 1,2,3, get a score/loss/

(6)Use score/loss get from the above 4 steps, gets an average of the score/loss and use it for metric for the model we are trying to evaluate.

So, in a k-fold cross validation we need to train the model k times and get the average of k scores/losses as the metrics. In real scenario, the number of k is always 5, also 3,4 or 10 is also used.

In my program, the k for the cross-validation is set to 5, which means the training Data will be split into 5 equal parts and 4 parts will be used for training while the remaining one will be used for validation, and the whole process will be repeated 5 times.

2.3.3 RandomizedSearchCV And GridSearchCV

There are 2 most important methods of searching for hyperparameters,which are gridsearchcv and randomizedsearchcv.

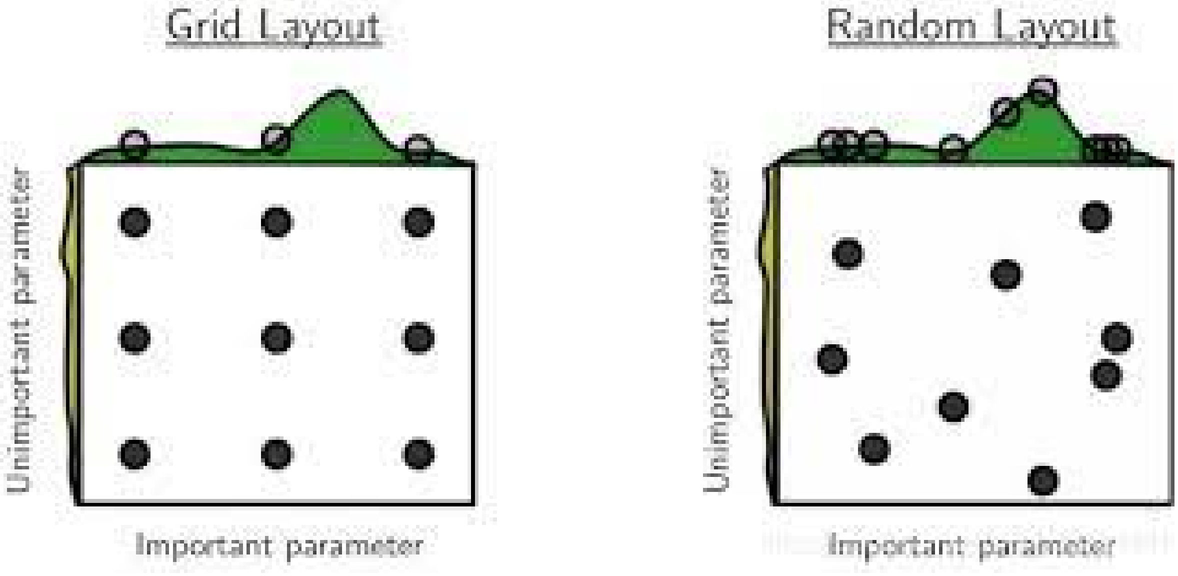


Figure 2.11: grid search and random search [3]

The grid search goes through all the possible combination of hyperparameters one by one while random search randomly selects some combination of hyperparameters. When the training data size increases and combination of hyperparameters is large, the time spent by grid search will be very long. In this case using random search will be better.

A randomize search did the following:

1.Get the parameter distribution for hyperparameters

2.Use k-fold cross validation for models with different combination of hyperparameters that are randomly selected

3.Get the hyperparameters that has the best performance

A grid search did the following: 1.Get the parameter combinations for hyperparameters 2.Use k-fold cross validation for models with all possible different combination of hyperparameters that are randomly selected 3.Get the hyperparameters that has the best average score/loss

For example, for the SVC model I am using, it has 2 hyperparameters: ”C”:[1e-2,1e-1,1,10,100],”gamma”:[1e-3,1e-2,1e-1,1,10] Which are C and gamma

In a grid search, we can see there are 5 possible values of C, and 5 possible values of gamma, so possible combination of hyperparameters will be 5*5 = 25.

The grid search will build 25 different kinds of models using each combination. For each single model, it will be evaluated using 5-fold cross validation to get an average score.

The scores from these 25 models will be compared, the combinations which has the best score will be selected as the best hyperparameter. This is what grid searchcv do.

In this process we trained the model 25 * 5= 125 times. Imagine we are training a model which has 3 or 4 kinds of different hyperparameters, Then the number of possible combinations can be really large, like millions or even Billions. Using gridsearchcv will be really time consuming.

In this case, randomizedsearchcv can be used for our target. Randomizedsearchcv does not go through all the combinations. Instead, it will randomly select some combinations.

If you set the number of combinations to 50, no matter how many combinations the hyperparameters will have , the randomizedsearchcv will only select 50 combinations and use them to build model. This can save a lot of time, also, in exchange of best performance.

Chapter 3

Methodology

3.1 Experimental Setup

The server used for the experiment is a ThinkBook laptop running window10. It has eight 11_{th} Gen Intel(R) (2.80GHz, Core (TM)) processors, and a 16GB RAM. The experiment is conducted in jupyter notebook 6.3.0 with python version 3.8.

3.2 Dataset

The dataset is directed imported from keras, which is a tensorflow library. The dataset has a training set of 60,000 examples and 60,000 labels And a test set of 10,000 examples and 10,000 labels.[1]

The example is a 28 * 28 matrix, the element of matrix representing The grayscale in the 28 * 28 image. The label is a number which ranges from 0 to 9, which is the handwritten number in the corresponding example.

The dataset is a combination from the handwritten digits written by American Census Bureau employees and American high school students. It has only black and white color so the image is $28 * 28$ rather than $28 * 28 * 3$.[2]

n order to make the dataset usable for training, the $28 * 28$ matrix will be flattened into an array of size 784, in which each element of an array represents a new feature.

Then the elements of data needed to be scaled. As K-neighbors classifier and Support vector classifier are sensitive to the range of features, all elements need to be rescaled into $[0, 1]$

$$pixel = pixel/255 \tag{6}$$

In which pixel represents the elements in training data and test data. As the largest possible value for pixel is 255, dividing the pixel by 255 will make the new range $[0, 1]$

For the convolutional neuron network, the input need to be reshaped into a 28 * 28 * 1 3D matrix in order to be used as the input.

3.3 Metrics

The occurrence of different labels in the training set is shown in Fig 3.1

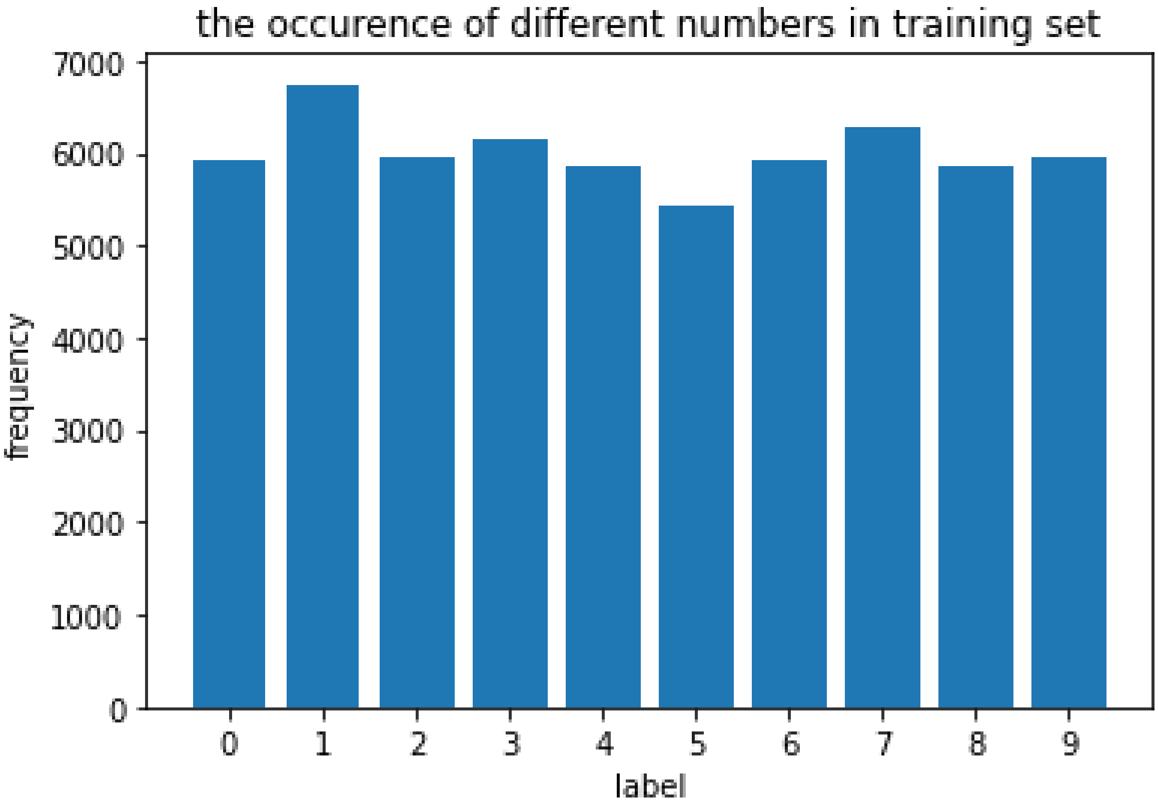


Figure 3.1: The occurrence of different numbers in training set

We can see that the frequency of different number is relatively balanced, with the lowest frequency over 80% of the highest frequency, so accuracy can be used as the metric for classification.

3.4 PCA And Standardization Of Inputs

The support vector classifier and K-neighbors classifier are sensitive to input features. In order to improve the performance PCA decomposition is applied to training dataset.

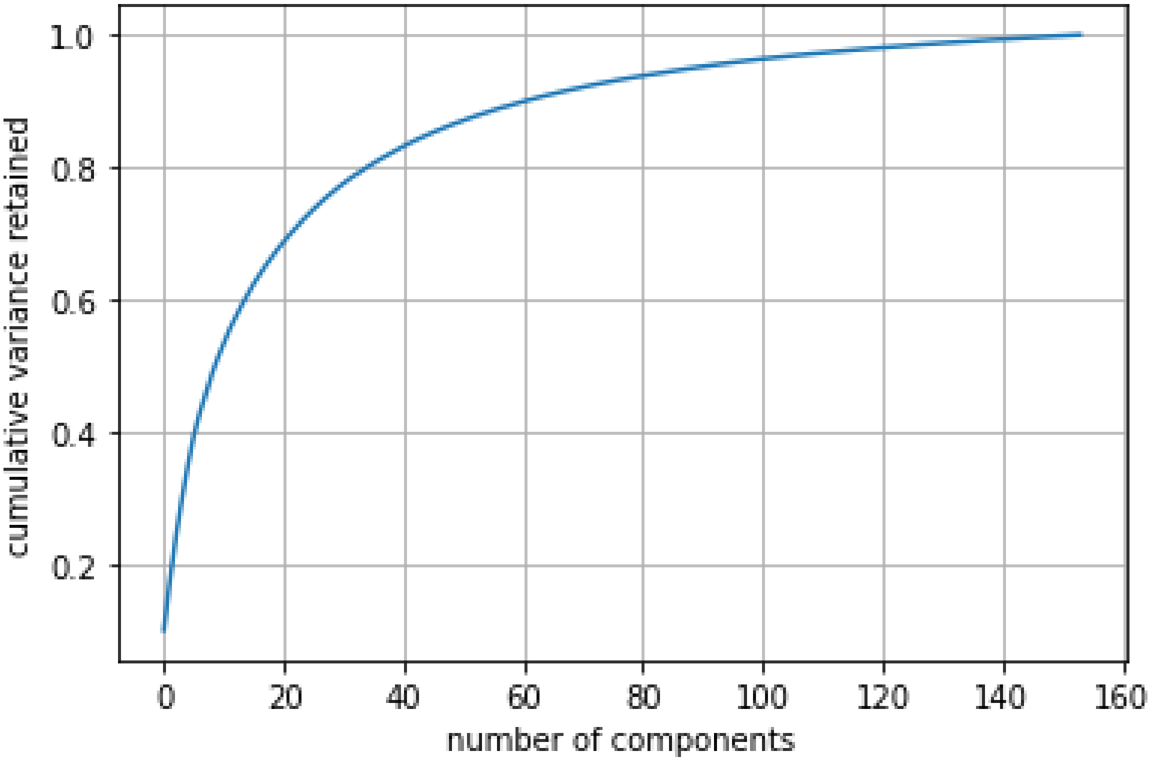


Figure 3.2: The cumulative variance of components

The number of components after dimensional reduction is not appointed. Besides, A threshold of 95% , which is the percentage of the variance kept by the components after processing, is appointed.

In Fig 3.2, it can be shown that the cumulative variance increases as the components increase, and the speed of increasing is becoming much slower while the components Become larger than 100.

In order to keep the overall variance of 95%, at least 154 components are needed for PCA. So by using pca on the training dataset a new training set of $6,0000 * 154$ is available.

The input of support vector classifier and k-neighbors classifier requires rescaling, so the new training set will be standardized by using the following:

$$val = (val\text{–}mean)/std \tag{7}$$

in which *mean* is the average value of the new component and *std* is the standard deviation of the new component.

By applying this, all new components will have a average value of 0 and standard deviation of 1. The standardized dataset will be used for SVC and KNN classifiers.

3.5 Constructing Convolutional Neuron Network

By directly using API in keras, I construct my cnn with 3 convolutional layer, 3 maxpooling layer, 1 flatten layer, and 1 dense layer.

The setup for layers is in table 3.1

Layers	setup
Conv2D	filters=32, kernel_size=3,activation=relu,padding=same
MaxPool2D	pool_size=(2,2)
Conv2D	filters=64, kernel_size=3,activation=relu,padding=same
MaxPool2D	pool_size=(2,2)
Conv2D	filters=128, kernel_size=3,activation=relu,padding=same
MaxPool2D	pool_size=(2,2)
Flattern	
Dense	unit=10,activation=softmax

Table 3.1: The setup of layers in cnn

The setup for training is in table 3.2

hyperparameter	setup
loss	Sparse_categorical_crossentropy
optimizer	adam
metrics	accuracy
epochs	20
validation_split	0.25

Table 3.2: training setup

In order to simplify my model I do not add dropout Layer. For the output is a Single digit the loss function is "Sparse_categorical_crossentropy".

Chapter 4

Models and Performance

4.1 Cross Validation

4.1.1 K-Neighbors Classifier

As the laptop itself cannot support using all 60,000 samples for cross validation. 10,000 samples are taken using StratifiedKFold in order to keep the ratio of the label unchanged in the shuffling data. The setup for cross validation is in Table 4.1

Setup	Property
cv folds	5
hyperparameters	{"n_neighbors":range(1,12)}
scoring	accuracy

Table 4.1: The setup of the cross validation for knn

The performance of different hyperparameters is shown in Fig 4.1

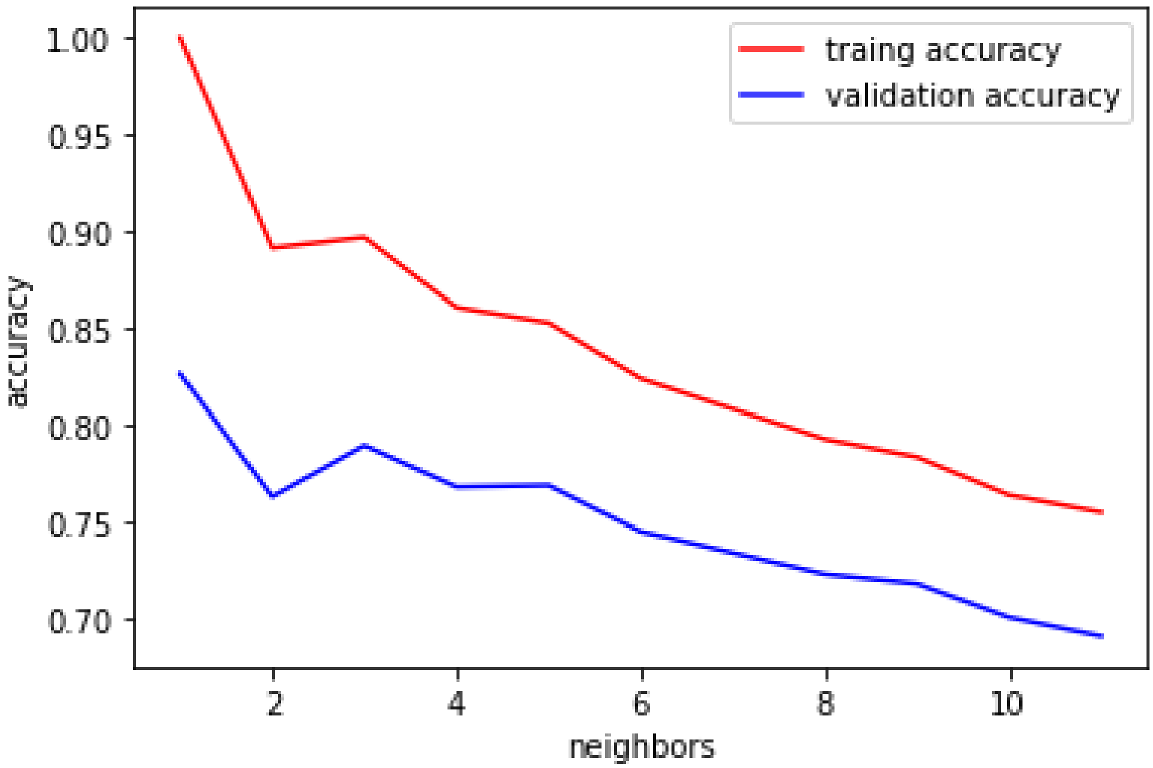


Figure 4.1: Score of different hyperparameters for knn classifier

As is shown in Fig 4.1, while n_neighbors = 1, both the training and validation has the highest performance, and as the number of n_neighbors increases, the performance on both training set and validation set becomes worse, which shows that the model is not learning. So n_neighbors = 1 will be the selected hyperparameter for k-neighbors classifier.

4.1.2 Support Vector Classifier

Similar to knn classifier, 10,000 samples are used in the validation of support vector classifier. The setup for cross validation is in Table 4.2

Setup	Property
cv folds	5
hyperparameters	{”C”:[1e-2,1e-1,1,10,100],”gamma”:[1e-3,1e-2,1e-1,1,10]}
scoring	accuracy
combinations	10

Table 4.2: The setup of the cross validation for svc classifier

For svc classifier, there are 2 kinds of hyperparameters. In order to reduce the time on cross validation, random search is performed, which means not all combinations of hyperparameters will be tried.

The performance of different hyperparameters is shown in Fig 4.2

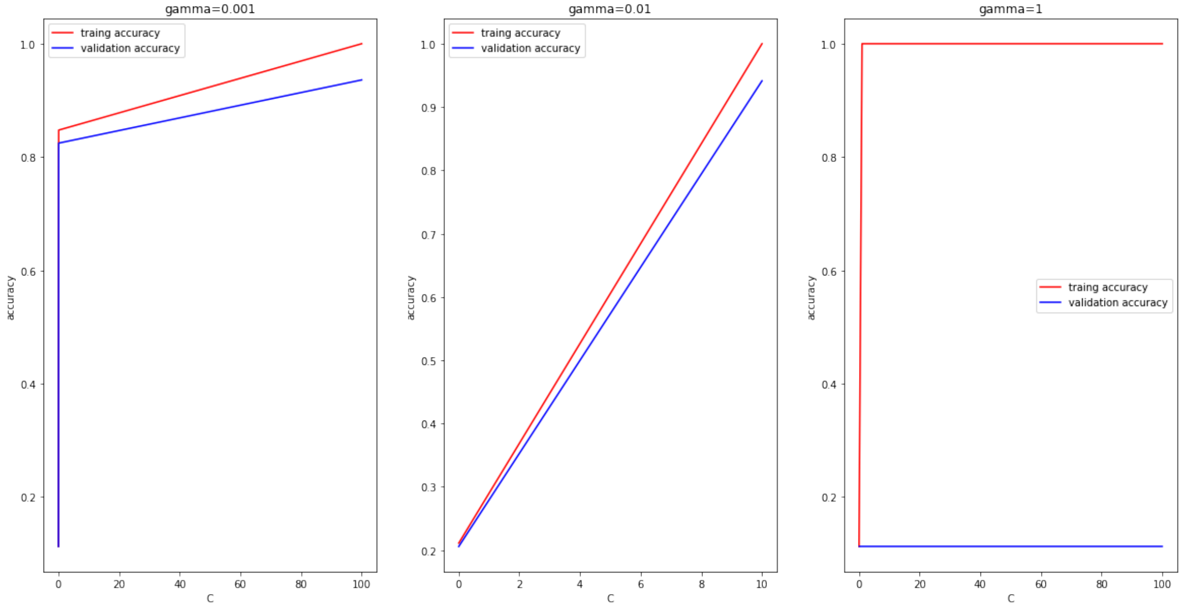


Figure 4.2: Score of different hyperparameters for svc classifier

It can be concluded from Fig 4.2 that while gamma = 1, the performance on training data is well with nearly 100% accuracy while it performs really bad on validation set with an accuracy of 11%. The performance of model while gamma = 0.01 is much better with 94% accuracy on validation sets. In this scenario, the best hyperparameters for svc will be gamma=0.01 and C=10

4.1.3 Decision Tree

For Decision Tree only max_depth is considered as the hyperparameter thatrequires tunning. In this case grid search is applied for going through all possible values.

The setup for cross validation is in Table 4.3

Setup	Property
cv folds	5
hyperparameters	{"max_depth": range(5,20)}
scoring	accuracy

Table 4.3: The setup of the cross validation for decision tree

The performance of different hyperparameters is shown in Fig 4.3

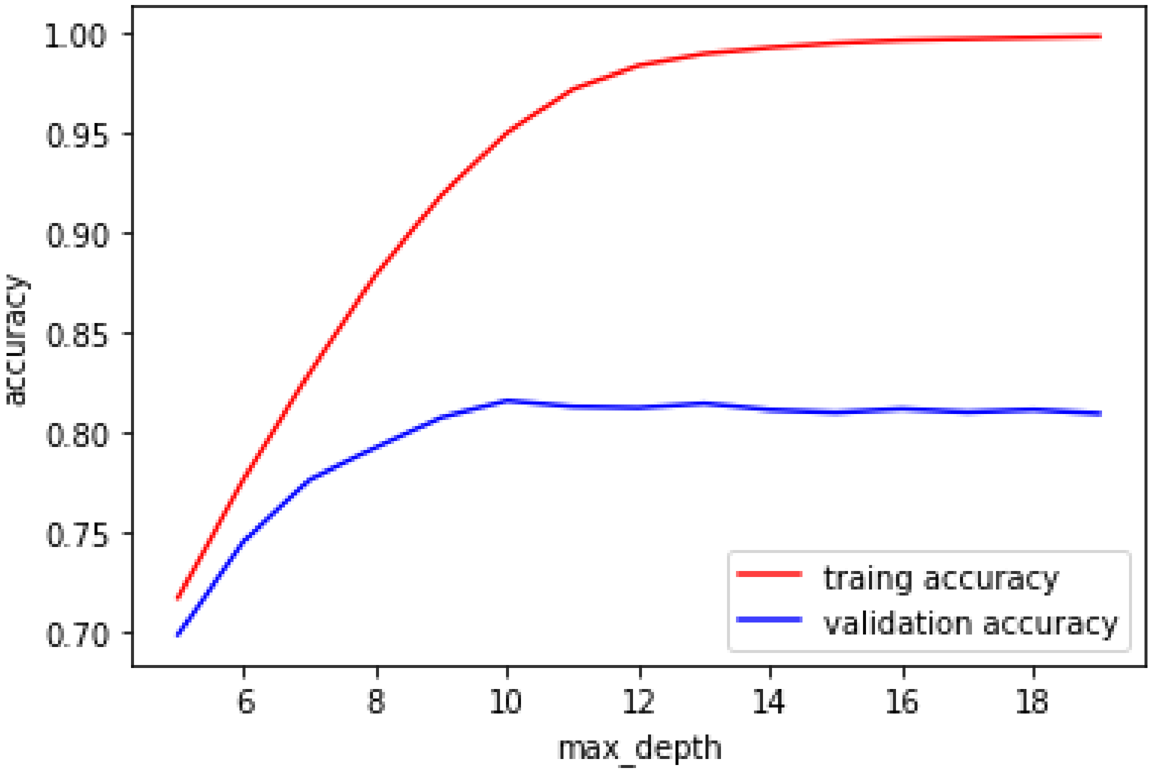


Figure 4.3: Score of different hyperparameters for decision tree classifier

From Fig 4.3 it can be seen that the accuracy increases as the depth is smaller than

10, after which the training accuracy continue increases while validation accuracy decreases, which means overfitting. So In this case the best hyper-parameter for decision tree will be max_depth = 10

4.1.4 Random Forest

For Random Forest the n_estimators is considered as the hyperparameter that requires tuning, for other hyperparameters are just the same as the decision tree. In this case grid search is applied for going through all possible values.

The setup for cross validation is in Table 4.4

Setup	Property
cv folds	5
hyperparameters	{"n_estimators": np.arange(10,110,10)}
scoring	accuracy

Table 4.4: The setup of the cross validation for decision tree

The performance of different hyperparameters is shown in Fig 4.4

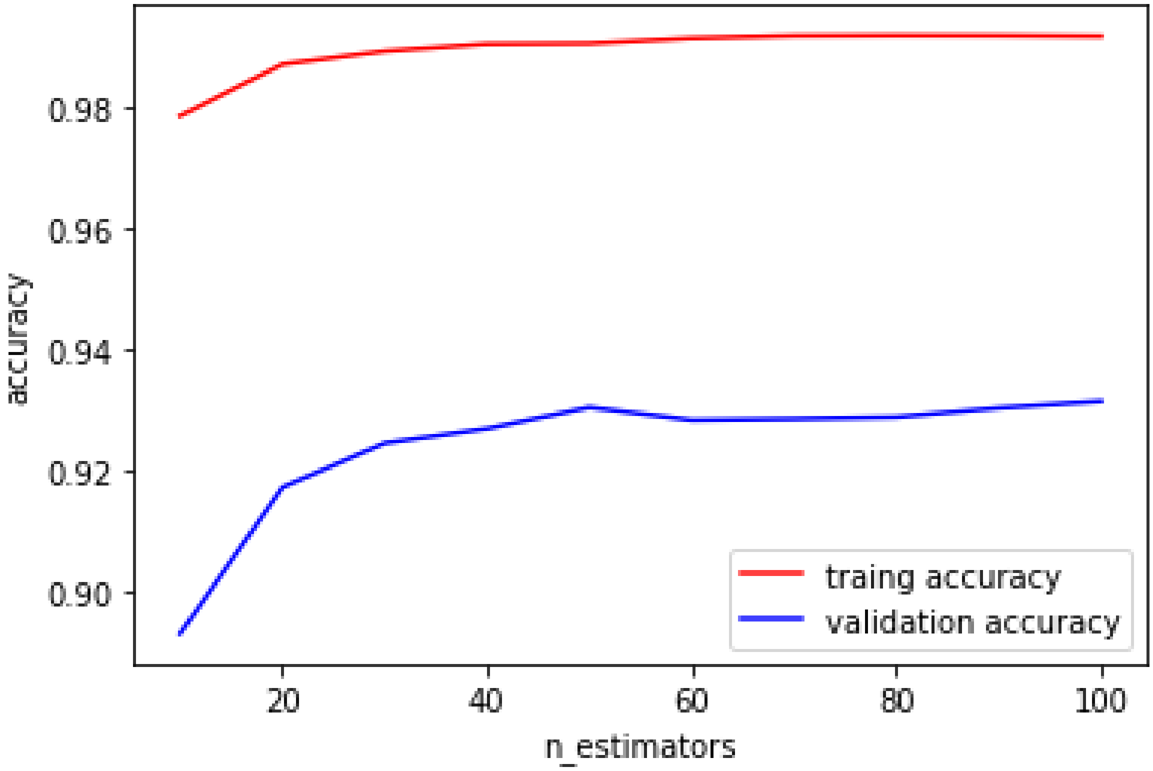


Figure 4.4: Score of different hyperparameters for Random Forest classifier

From Fig 4.4 it can be concluded that both training and validation accuracy increases as the number of estimators goes up. Although after n_estimators¿40 the increase is flat but the variance does not become larger, so n_estimators =100 will be the best hyperparameters for the Random Forest Classifier.

4.1.5 CNN

Unlike the traditional classifiers it will be so much work if I use k-fold cross validation, So I just train the models on the whole training set but set a parameter validation_split, Which determines the proportion of the data used in the validation. In my network, The validation_split is set to 0.25, which means 75 % of the training data will be used for Training while the remaining 25% will be used for validation in each epoch.

The loss and accuracy through training is in Fig 4.5

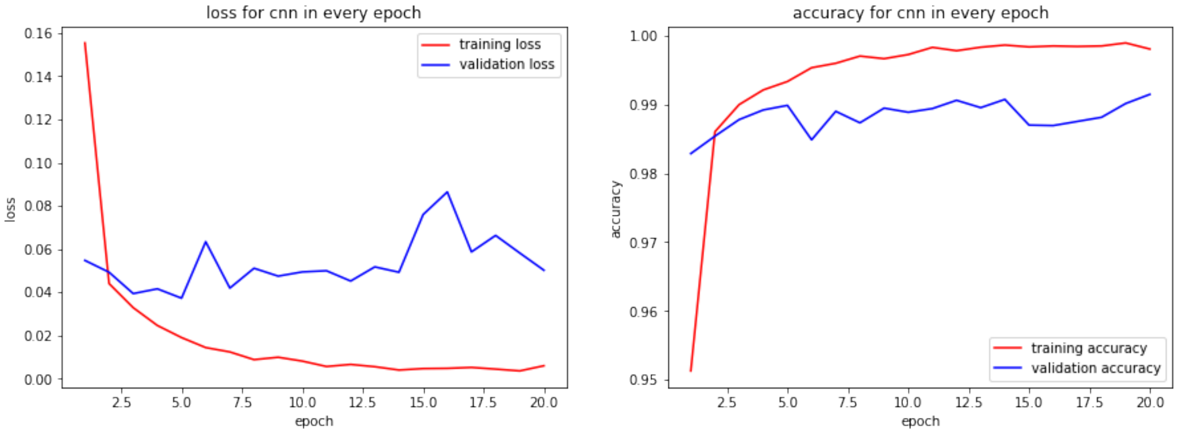


Figure 4.5: The loss and accuracy for training and validation set in each epoch

From the figure we can see that the training loss continues to decrease after each epoch, But the validation loss/accuracy did not improve after epoch = 3. But the validation loss does not become bigger, which means the model is not overfitting even the number of epoch is large. So the models trained after 20 epoches will be used for final prediction.

4.2 Performance

Five traditional classifiers, knn classifier, svc classifier, decision tree classifier, random forest classifier ,GaussianNB and the convolutional neuron network are used for predicting the test data.

The svc and knn requires the input to be preprocessed first, so their input had gone through pca dimensional reduction and standardization before being passed into the model.

All the models used 60,000 examples for training. The performance of models on test data is shown in Fig 4.6

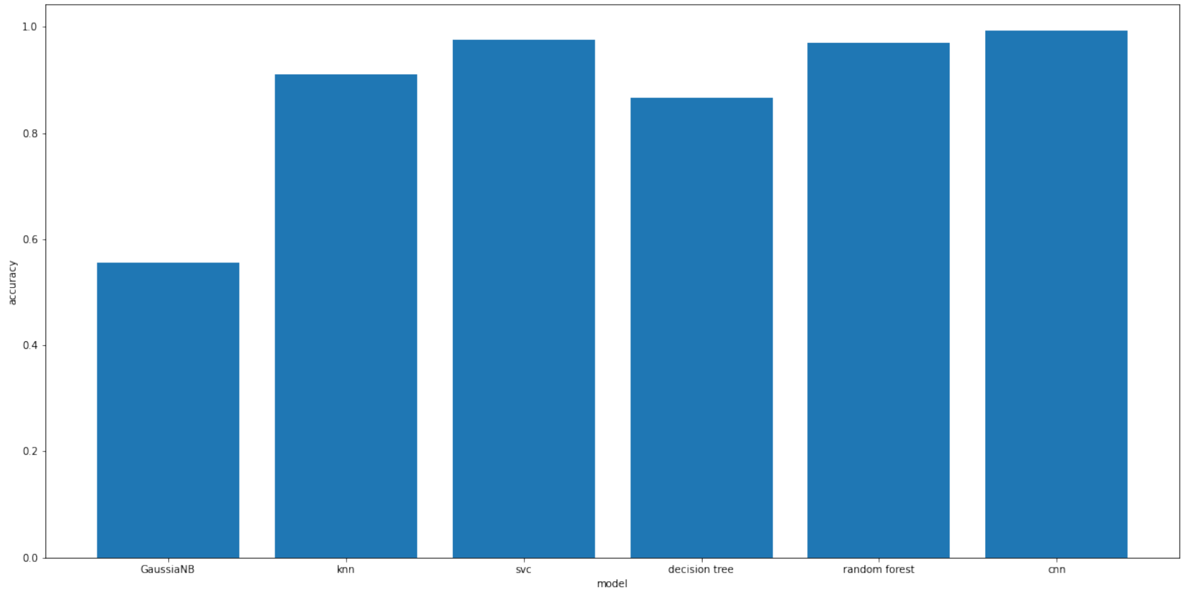


Figure 4.6: the performance of models on test data

From the Fig 4.5 it can be seen that cnn and svc has the top performance. With 99.26 % accuracy for cnn and 97.58% accuracy for random forest classifier.

The GaussianNB has the worst performance of only 55.58%, which shows that Naïve bayes is not a good model in this scenario.

Chapter 5

Conclusion

In this report I worked on using different classification techniques to predict the image class on unseen data. Six different kinds of models are used in the experiment, which are GaussianNB, K Neighbors classifier, support vector classifier, decision tree classifier ,random forest classifier, and Convolutional Neuron Network. Dimensional reduction and standardization are applied so that the knn and svc models can work effectively. The GridSearchCV and RandomizedSearchCV are used for cross validation in order to find the best hyperparameters for the models. The performance on test data shows that cnn and Support Vector Classifiers are good at classifying the images while Naïve Bayes Classifiers are not suitable.

Bibliography

[1] https://www.datacamp.com/community/tutorials/k-nearest-neighbor- classification-scikit-learn.

[2] https://www.datacamp.com/community/tutorials/k-nearest-neighbor- classification-scikit-learn.

[3] https://blog.csdn.net/xiaohutong1991/article/details/107946291.

[4] http://yann.lecun.com/exdb/mnist/.

[5] https://en.wikipedia.org/wiki/MNIST_database.

[6] A. Palvanov and Y. Im Cho. Comparisons of deep learning algorithms for mnist in real-time environment. *International Journal of Fuzzy Logic and Intelligent Systems*, 18(2):126–134, 2018.