Medical Image Analysis Assignment 1

Jingmo Bai

Introduction

In this assignment we have three problems about different algorithms of Machine Learning.

The first is non-parametric estimation of probability density function, i.e., the parzen window Kernel estimation method. We need to apply this method to estimate the pdf from 20 training data of 2 classes normal distributed data points. And then use estimated pdf to classify the test data. In further we will study the influence of different kernel width to the estimation and apply cross-validation to estimate the optimal width.

The second is using hand-crafted features and different kinds of classifier to classify 6 types of cell images in HEP2 dataset, we need to find out the morphologic difference between different types of cells and design suitable features for them. And we need to compare different kinds of classifier and tune the hyper parameters of them.

The third is to build a CNN deep learning network to classify digits images and HEP2 cell images. We are going to study the architecture of deep neural networks and build several layers of network to do the classification.

Description

Kernel density estimation:

KDE is also called parzen window method, it uses different kernels and known data points to estimate probability density function. In a word, it weights the distance of all the data points for every sampling points. If there are more data points nearby, then the probability here is high. Different kernels decide how the points are weighted, we use gaussian kernel in this assignment.

Cross-validation:

Cross-validation is a technique to partition the data set into different portions to test and train with different portions every iteration. It trains the model on the training part and tests it on the validation part, the validation part is unseen and not used while training. So, it examines the model's ability to generalize on the future data.

K-Nearest Neighbor:

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

Random Forests:

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

Support Vector Machines:

SVM maps training examples to points in space so as to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Neural Network:

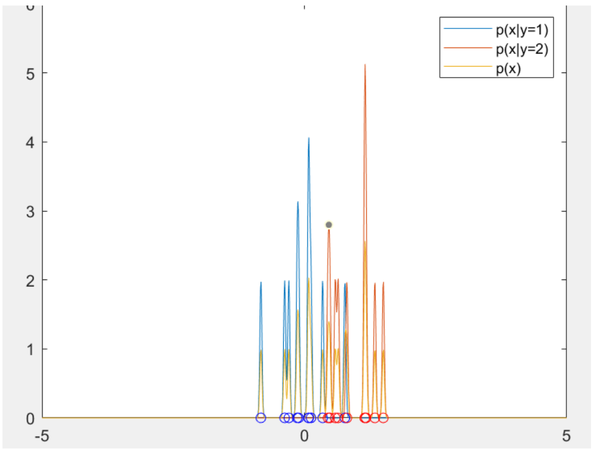
Neural Network is a algorithm that creates several layers of nodes from the input to the output. By assigning different weights and bias to the nodes and minimizing the loss function to our goal, it gets a combination of optimal weights to achieve different types of tasks.

Results

1. Kernel Estimation

1. For model standard deviation s_model = 0.4

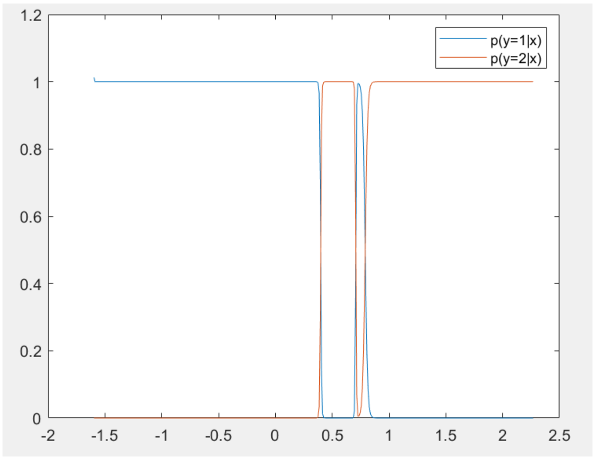
For kernel width s_parzen = 0.02



p(x | y = 1), p(x | y = 2), p(x)

For s_model = 0.4, s_parzen = 0.02

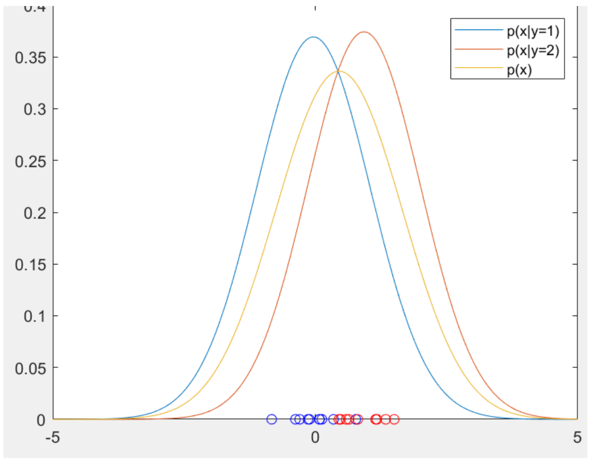
The estimated density is very squiggly and could not reshape the true normal distribution density function.



p(y = 1 | x), p(y = 2 | x)

For s_model = 0.4, s_parzen = 0.02

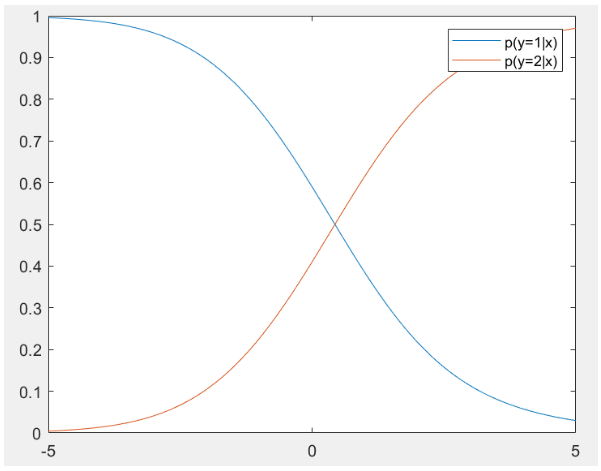
For kernel width s_parzen = 1



p(x|y = 1), p(x|y = 2), p(x),

For s_model = 0.4, s_parzen = 1

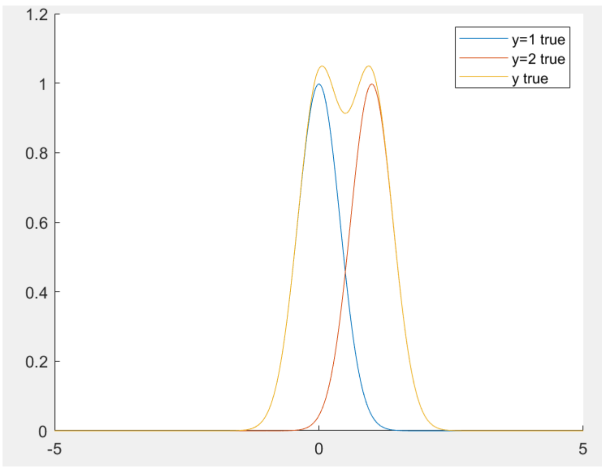
The estimated densities are much wider than the true models, but they are smooth enough to reshape the normal distribution, also the means are correct.



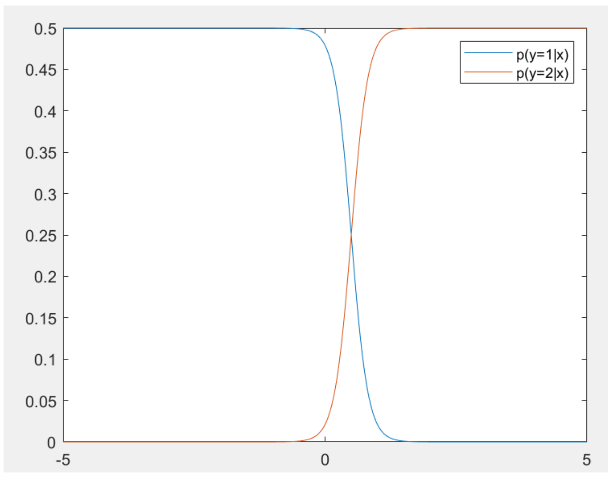
p(y = 1|x), p(y = 2|x)

For s_model = 0.4, s_parzen = 1

True model:



p(x|y = 1), p(x|y = 2), p(x)



p(y = 1|x), p(y = 2|x)

Using the estimated probability to classify the test data.

s_parzen = 0.02:

For class 1, 1000 test data, 859 are right and 141 are wrong.

For class 2, 1000 test data, 863 are right and 137 are wrong.

Error rate = 13.9%

s_parzen = 1:

For class 1, 1000 test data, 850 are right and 150 are wrong.

For class 2, 1000 test data, 921 are right and 79 are wrong.

Error rate = 11.45%

For true model, the optimal threshold will be 0.5, if less than 0.5 then class 1, else class 2.

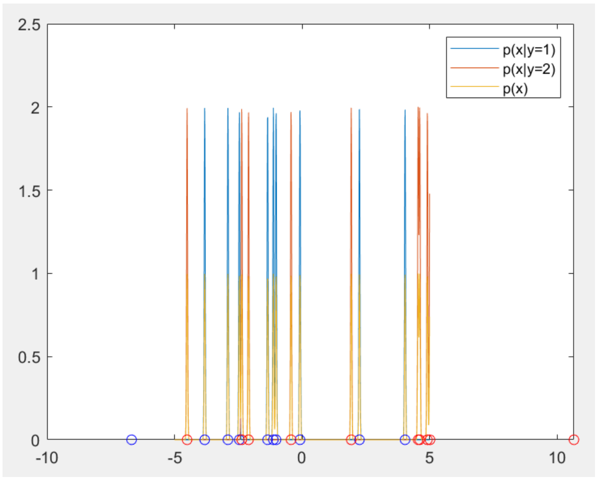
Find the cumulative distribution value when x =0.5, cdf (y=1) = 0.895, cdf (y=2) = 0.105

So we get the theoretical optimal error rate = 10.5%

This theoretical optimal error rate is slightly better than the estimated model with width 1, and fairly better than the estimated model with width 0.02.

2. For model standard deviation s_model = 4

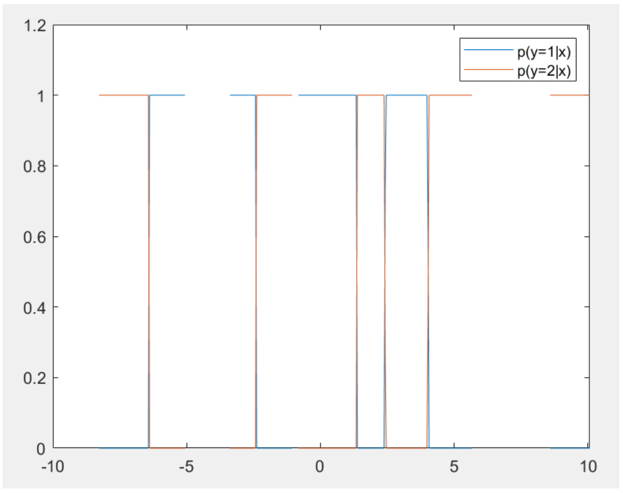
For kernel width s_parzen = 0.02



p(x|y = 1), p(x|y = 2), p(x)

For s_model = 4, s_parzen = 0.02

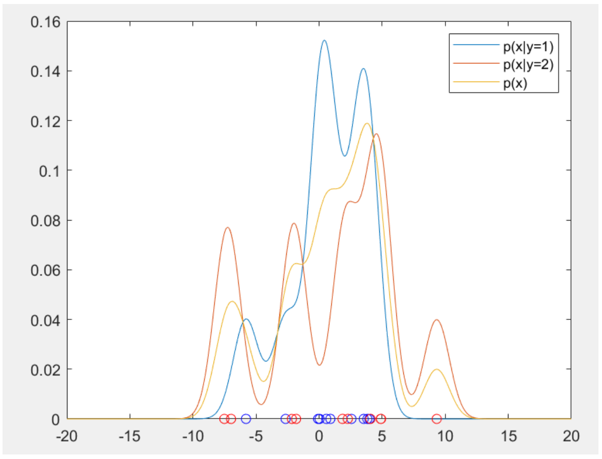
The estimated densities are very squiggly and could not reshape the true normal distribution density function.



p(y = 1|x), p(y = 2|x)

For s_model = 4, s_parzen = 0.02

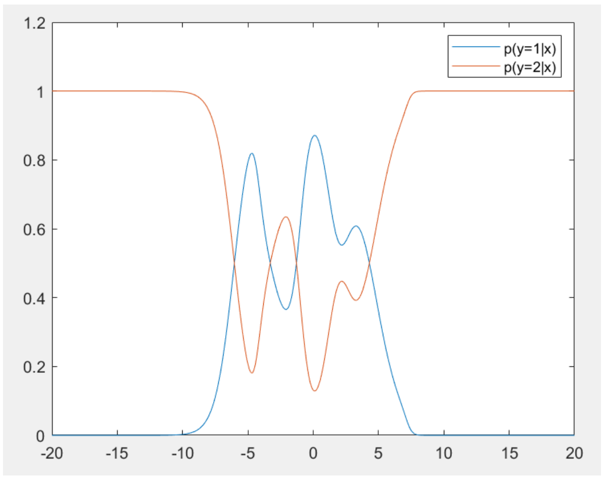
For kernel width s_parzen = 1



p(x|y = 1), p(x|y = 2), p(x),

For s_model = 4, s_parzen = 1

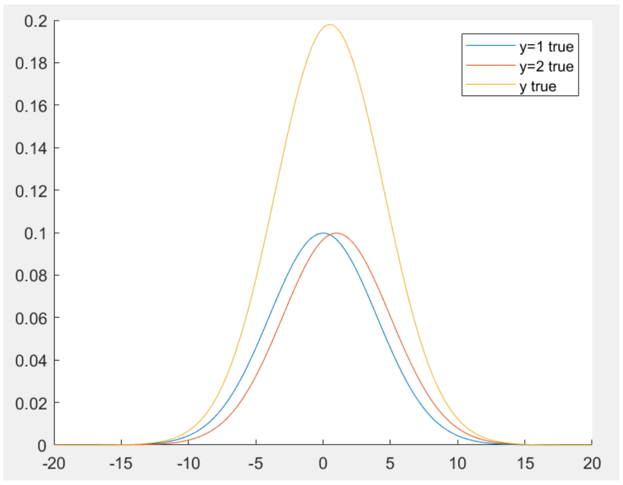
The estimated densities are smooth enough but not able to reshape the true models, because the training data is too little and the standard deviation is too large.



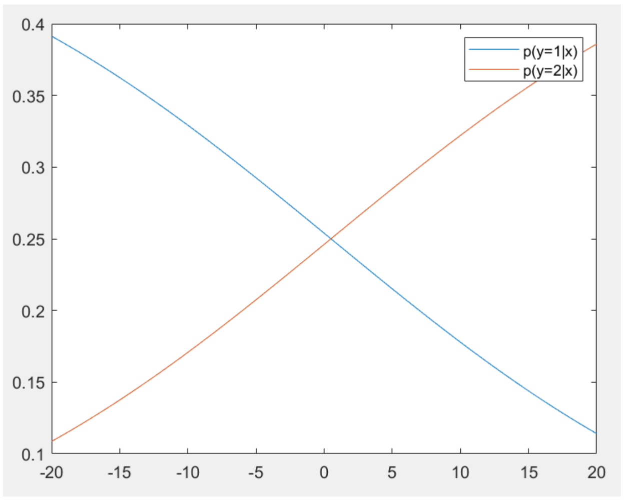
p(y = 1|x), p(y = 2|x)

For s_model = 4, s_parzen = 1

True model:



p(x|y = 1), p(x|y = 2), p(x)



p(y = 1|x), p(y = 2|x)

Using the estimated probability to classify the test data.

s_parzen = 0.02:

For class 1, 1000 test data, 461 are right and 539 are wrong.

For class 2, 1000 test data, 334 are right and 666 are wrong.

Error rate = 60.25%

s_parzen = 1:

For class 1, 1000 test data, 635 are right and 365 are wrong.

For class 2, 1000 test data, 373 are right and 627 are wrong.

Error rate = 49.6%

For true model, the optimal threshold will be 0.5, if less than 0.5 then class 1, else class 2.

Find the cumulative distribution value when x =0.5, cdf (y=1) = 0.55, cdf (y=2) = 0.45

So we get the theoretical optimal error rate = 45%

This theoretical optimal error rate is a little better than the estimated model with width 1, and much better than the estimated model with width 0.02.

Conclusion

The comparisons between model densities and estimated densities (s_model=0.4):

For kernel width = 0.02, i.e., the width is too low, the estimated densities are very squiggly and could not reshape the true normal distribution density function, because the window is too narrow so only points very close to the sampling points are given weight, while the number of training data is too little.

For kernel width = 1, i.e., the width is too high, the estimated densities are much wider than the true models, which means the means are estimated correctly but the standard deviations are estimated larger. This is because even distant points are given weight to form the density.

According to the plots of p(y=1|x) from estimated model and true model, the error in the estimated does have effect on the density of p(y=1|x), but the estimated model still manage to classify the data with a decent accuracy rate.

The comparisons between nearest neighbor and plug in classifier when s_parzen is low:

The working principle of nearest neighbor classifier is quite similar to when we use a small kernel width. The nearest neighbor classifier is just finding the closest value and return the same class. When kernel width is very low, the kernel only assigns weight to the training points very close to the sampling points, which is the same thing as picking the nearest neighbor. When we classify with this density, it just classifies according to the closest training data point. So they are quite similar.

In the equation of parzen density estimation:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K(\frac{x - x_i}{h})$$

When width h⟶0, and x = xi, $\frac{x-x_i}{h}$ = 0, K($\frac{x-x_i}{h}$) $= 1$;

When x≠xi, $\frac{x-x_i}{h}$ ⟶∞, K($\frac{x-x_i}{h}$) ⟶ 0;

So in this case, only when x = xi, the estimation density will have high value, and the density will decrease rapidly when moving away from sampling points. Using this to do classification equals to choosing the nearest points to the sampling points, the nearer to the points, the higher the estimated density will be. So it becomes 1-NN classification.

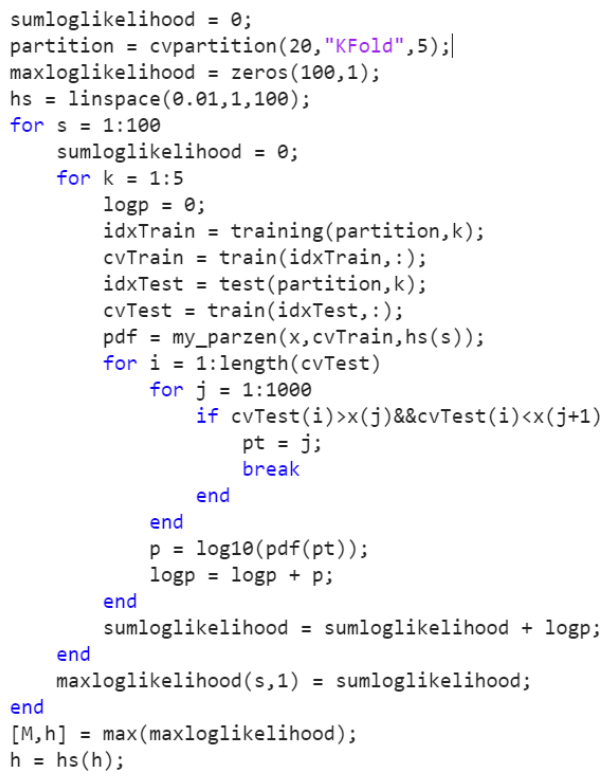
Cross-validation to estimate kernel width:

First, I use "cvpartition" to partition the training dataset, I use K-fold and k is 5. So there are 5 sets in total. Every time, we pick one of them as test data and all the rest 4 of them as training data. Each time we use the training data set to estimate the density, and use this estimated density to plug-in the test data. After plugging in, we use the log-likelihood and sum them up. In the end, we need to maximize the sum to find the optimal kernel width.

We do it in a for loop, where we try out 100 values of width in the range of 0.01 to 5. With every value of width, we calculate a sum of log-likelihood, and we finally get a maximum log-likelihood and its width. This width is the optimal width.

For s_model=0.4, the optimal s_parzen turns out to be around 0.4.

For s_model=4, the optimal s_parzen turns out to be around 3.8.



```
sumloglikelihood = 0;
partition = cvpartition(20,"KFold",5);
maxloglikelihood = zeros(100,1);
hs = linspace(0.01,1,100);
for s = 1:100
    sumloglikelihood = 0;
    for k = 1:5
        logp = 0;
        idxTrain = training(partition,k);
        cvTrain = train(idxTrain,:);
        idxTest = test(partition,k);
        cvTest = train(idxTest,:);
        pdf = my_parzen(x,cvTrain,hs(s));
        for i = 1:length(cvTest)
            for j = 1:1000
                if cvTest(i)>x(j)&&cvTest(i)<x(j+1)
                    pt = j;
                    break
                end
            end
            p = log10(pdf(pt));
            logp = logp + p;
        end
        sumloglikelihood = sumloglikelihood + logp;
    end
    maxloglikelihood(s,1) = sumloglikelihood;
end
[M,h] = max(maxloglikelihood);
h = hs(h);
```

2. Cell type classification

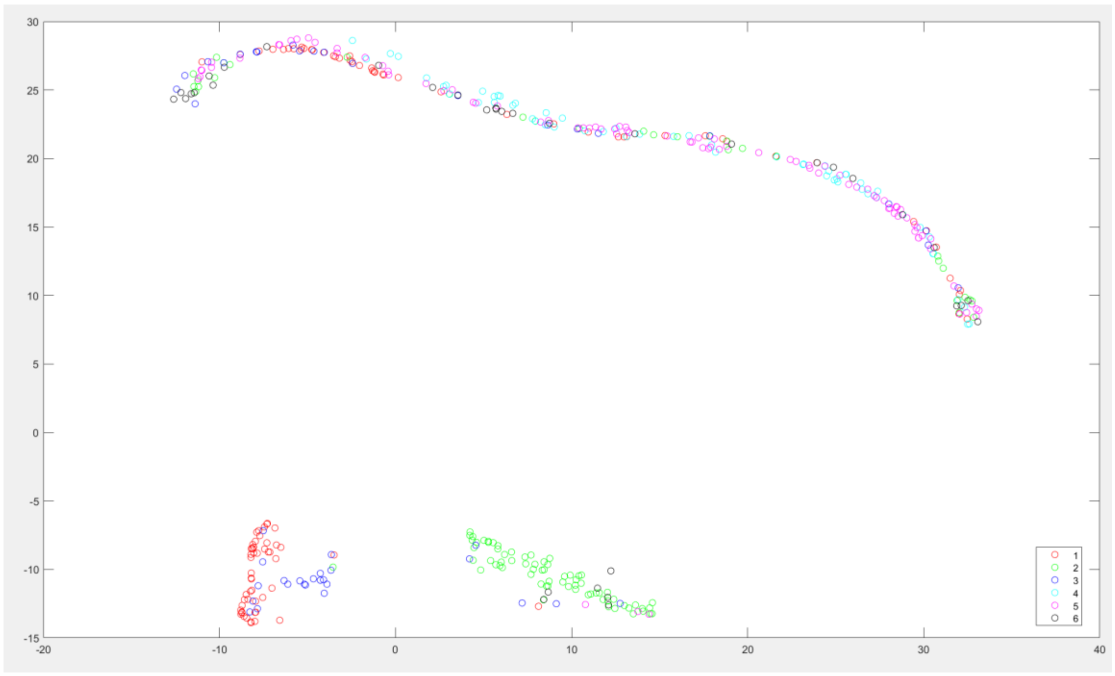
The features I tried:

First, I apply the edge detection canny filter to the original images, to obtain the edge information of the cell.

Then I extract several features from the filtered images, some regions properties like “Area”, which is the actual number of pixels in the region. “Extent”, which is the ratio of pixels in the region to pixels in the total bounding box. “Filled Area”, which is number of pixels when all holes filled. “Euler Number”, which is number of objects in the region minus the number of holes in those objects.

Eventually I use 6 features in total, the four ones above plus mean and standard deviation of my edge mask.

Visualize:



From the visualization, we can see that class 1, class 2 and class 3 can be separated in different regions in the feature space but the others classes are not well separated. It is hard to find suitable features by hand-coded.

Machine learning methods tried and accuracy:

	Training accuracy	Testing accuracy
Decision tree	0.87209	0.48037
Random forest	1	0.49192
SVM	0.65814	0.55196
KNN	0.82093	0.53349
Neural network	0.81628	0.50115

According to my best result of the accuracy on the test data, the performance on future data will be about 55% accuracy. I would recommend SVM or KNN models, they have the best generalization performance so far.

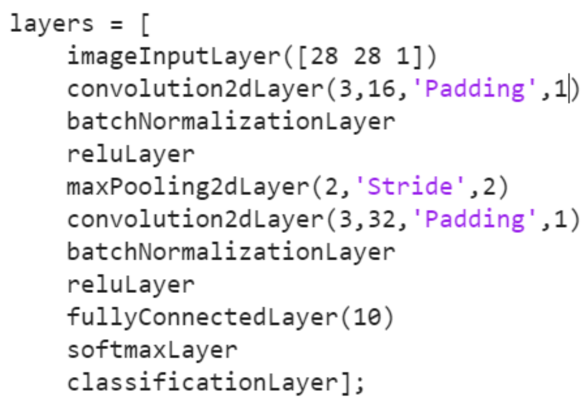
We can see that the accuracy on the training data is much higher than the accuracy on the test data, which implies the models are overfitting to the training data, and not very good at generalization. This is due to the difference and bias between different data sets, different medical instruments, etc. Our training data may often be biased and we have no access to more training data.

I have tried some techniques to improve the model. For example, I apply normalization to the features before training the data, it rescales the values to zero mean and unit stand deviation, which prevent one feature to be dominant during the training. I also use the optimization Hyperparameters options to train the models, which minimizes k-fold cross validation loss to determine the optimal hyperparameters for each model.

3. CNN

For digits dataset:

The network architecture I experimented with :



```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

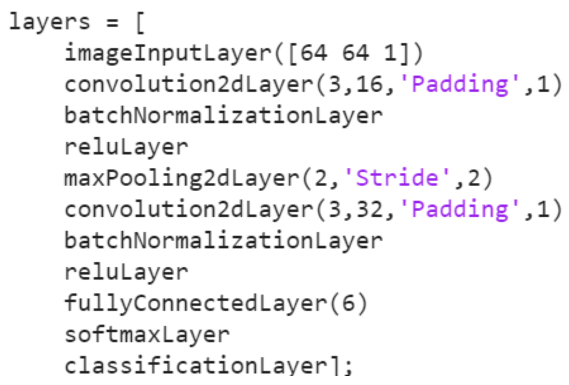
Table for training and testing accuracy of default architecture and deeper one:

	Training accuracy	Testing accuracy
Default architecture	0.8128	0.7092
Deeper architecture	0.988	0.969

For HEP2 dataset:

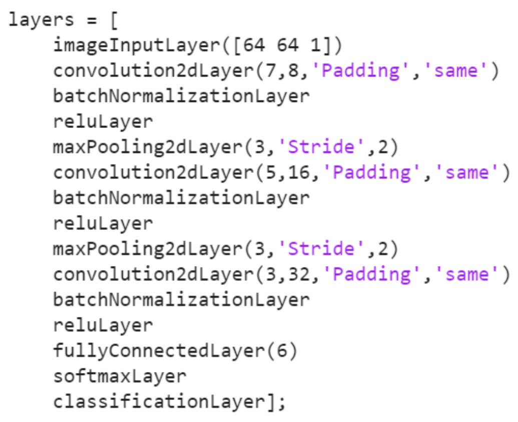
The network architecture I experimented with :

Network 1:



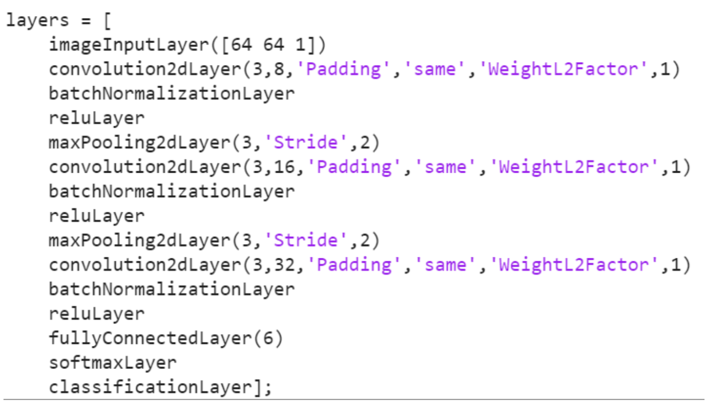
```
layers = [
    imageInputLayer([64 64 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(6)
    softmaxLayer
    classificationLayer];
```

Network 2:



```
layers = [
    imageInputLayer([64 64 1])
    convolution2dLayer(7,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(5,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(6)
    softmaxLayer
    classificationLayer];
```

Network 3:



```
layers = [
    imageInputLayer([64 64 1])
    convolution2dLayer(3,8,'Padding','same','WeightL2Factor',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(3,16,'Padding','same','WeightL2Factor',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(3,32,'Padding','same','WeightL2Factor',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(6)
    softmaxLayer
    classificationLayer];
```

Table for different architectures and accuracy:

	Training accuracy	Testing accuracy
Network 1	0.62326	0.32794
Network 2	0.98837	0.4157
Network 3	0.96512	0.54503

We can see the training accuracy is every high while the testing accuracy is much lower. This implies the models are overfitting to the training data and not able to generalize to future data. This may due to the difference and bias between different data sets, different medical instruments, etc. Our training data may often be biased because the source is limited.

I tried some data augmentation methods to try to improve the testing accuracy.
With the “imageDataAugmenter” function, we can configure a set of preprocessing options for image augmentation, such as resizing, rotation, and reflection. The principle is, because we only have limited data to train, we thus modify our training data in some ways artificially and make them more diverse. I tried these augmentations on the training dataset and it did improve the test accuracy, making the model more generalizable.

I also tried some other techniques which deals with overfitting like drop out, L1/L2 regularization, batch normalization. For drop out, it seems to make the accuracy worse. For regularization term, they did help a little. But the models are still to overfitting overall.

The initial learning rate is 0.01. I use ADAM algorithm. It computes adaptive learning rate to each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients. I also tried learning rate drop which will drop the learning rate every time a certain number of epochs passes, which prevents the initial learning rate is too large, leading to a local minimum.

The maximum epochs is 50. I find there is no need to increase it more, because after around 50 epochs, the model often can not improve any more, the accuracy on the mini batches are already very high, so there is no need to train longer.