汇报

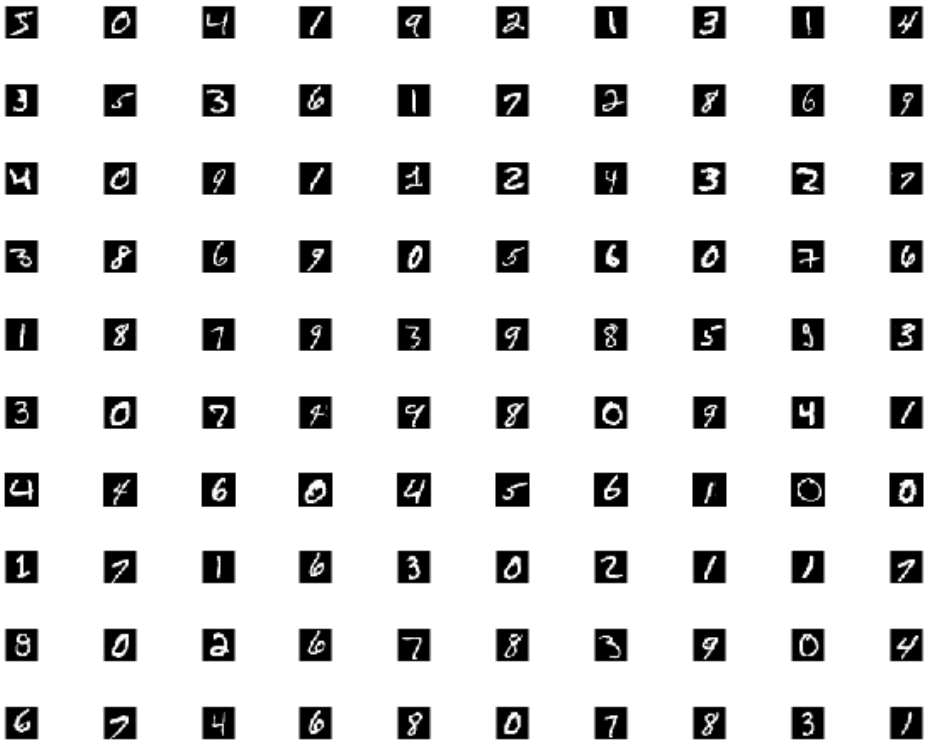
罗明昊-20338053

MNIST

请尝试对MNIST数据集来实现自己的一些方法。方法可以是课件里讲的模型的调整（如图像预处理、参数或结构调整），可以是通过文献或网络资料学习的方法（如决策树、随机森林、贝叶斯方法、循环神经网络等），也可以是自己设计的方法。将他们用任意语言实现后，写出你的实现方法和结果，并进行分析比对。

先查看MINST数据集。

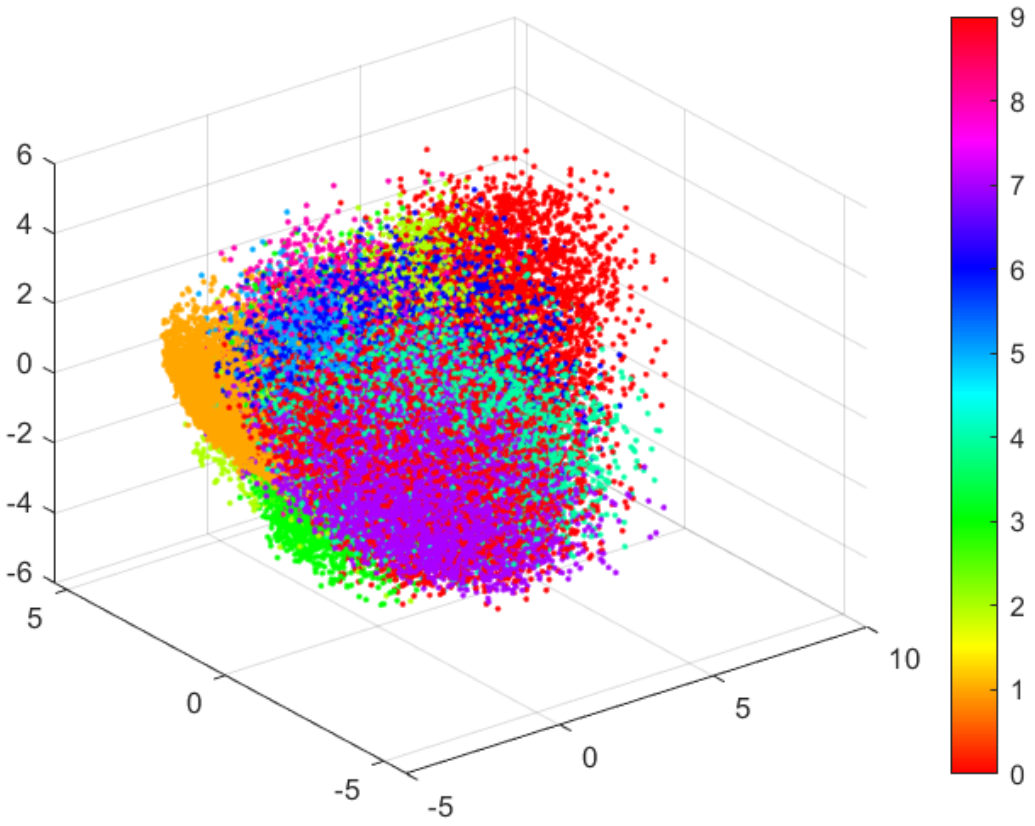
```
images = LoadMNISTImages('train-images.idx3-ubyte');
for i = 1:10
for j = 1:10
k = (i-1)*10+j;
subplot(10,10,k);
temp = reshape(images(:,k),28,28);
imshow(temp,[]);
end
end
```



Matlab

用主成分分析和降维方法图示训练集 96.60%

```
images = LoadMNISTImages('train-images.idx3-ubyte');
labels = LoadMNISTLabels('train-labels.idx1-ubyte');
images_centered = images - mean(images,2);
PCA_M = images_centered*images_centered';
[V,D] = eigs(PCA_M,784,'la');
images_coordinate = V'*images_centered;
images_PC = images_coordinate(1:3,:);
scatter3(images_PC(1,:),images_PC(2,:),images_PC(3,:),[],labels,'.');
colormap hsv,colorbar
```



说明相同的数字图往往距离较近，不同的数字图往往距离较远。对于MNIST数据集我们可以以测试集的第20幅图为例，通过计算欧式距离找出和该图相似的图像，通过返回 KNN_labels 对识别结果进行检验。

```
test_image = images_test(:,20);
temp1 = reshape(images_test(:,20),28,28);
imshow(temp1,[])
x_diff = images_train - test_image;
x_dist = sqrt(sum(x_diff.*x_diff));
[~,x_ranking] = sort(x_dist);
KNN_labels = labels_train(x_ranking(1:30));
KNN_labels'
GroundTruth_label = labels_test(20)
```

```
test_image = images_test(:,20);
temp1 = reshape(images_test(:,20),28,28);
imshow(temp1,[])
```

```
x_diff = images_train - test_image;
%计算训练集和该图像的差距
x_dist = sqrt(sum(x_diff.*x_diff));
%根据差距计算出欧氏距离
[~,x_ranking] = sort(x_dist);
%排序，无需返回排序后的序列，只返回排序后从小到大的列指标
KNN_labels = labels_train(x_ranking(1:30));
KNN_labels'
```

```
ans = 1x30
     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     9     4     4
```

```
GroundTruth_label = labels_test(20)
```

```
GroundTruth_label = 4
```

可以看出使用 KNN 方法判断出和原图像“距离最近”的30幅图绝大多数标签都与第20幅图一致，标签为4，而距离排28的图像识别出的标签是9，猜测是因为手写的9和4形状相似造成的，查看一下这幅图。

```
temp2 = reshape(images_train(:,x_ranking(28)),28,28);
imshow(temp2,[])
```

确实更接近9，但是和测试集的第20幅图也确实有相似的地方。

对 KNN 模式识别的方法进行评分并计算其正确率。

```
Correct_Count = 0;
for j = 1:10000
x_diff = images_train - images_test(:,j);
x_dist = sqrt(sum(x_diff.*x_diff));
[~,x_ranking] = sort(x_dist);
KNN_labels = labels_train(x_ranking(1:20));
%最近的20个标签不同时，需要做出合理的决策，如何综合利用KNN数据，为该类方法带来了很多的变化
temp = zeros(1,10);
for i = 1:20
temp(KNN_labels(i)+1) = temp(KNN_labels(i)+1) + (21-i);%一种粗糙评分，最近的评20分，第二近的19分，以此类推
end
[~,Final_label] = max(temp); %寻找得分最高的数字
Final_label = Final_label - 1; %不要忘记减1代表0-9
GroundTruth_label = labels_test(j);
```

```
if(Final_label == GroundTruth_label)
Correct_Count = Correct_Count + 1;
end
end
Correct_rate = Correct_Count/10000;
```

```
Correct_rate
```

```
Correct_rate = 0.9660
```

得到该方法的正确率大约为96.60%，产生的识别错误可能是因为上面提到的手写的部分数字确实在形状上较为相似，很难用距离来对它们进行区分。

最小二乘解 85.34%

尝试直接对训练集进行线性拟合，计算其最小二乘解，使得解尽可能满足方程组。

```
train_amount = 60000;
B = zeros(10,train_amount);
for i = 1:train_amount
B(labels_train(i)+1,i)=1;
end
AT = images_train(:,1:train_amount)';
X = (AT\B')';%直接获得最小二乘解
%% Test Data
Output_new = X*images_test;%得到10000组概率向量
[~,label_new] = max(Output_new);%获得每一组向量的最大下标所在位置
label_new = label_new -1; %位置-1即为预估的手写数字值
accu = labels_test==label_new';
sum(accu)/10000 %准确率的计算
```

```
sum(accu)/10000 %准确率的计算
```

```
ans = 0.8534
```

使用测试集对分类结果进行估计，发现其分类准确率约为85.34%，相比起 KNN 低一些，但是由于其运行速度较快，希望在短时间内进行粗糙分类的情况下更适合使用求最小二乘解的方法。

SVC 98.46%

使用支持向量机(SVM)用于分类。

```
images_train=LoadMNISTImages('train-images.idx3-ubyte');
images_test=LoadMNISTImages('t10k-images.idx3-ubyte');
labels_train=LoadMNISTLabels('train-labels.idx1-ubyte');
labels_test=LoadMNISTLabels('t10k-labels.idx1-ubyte');

% transpose 转置后，每一行为一幅图像 变为一个number*pixel的矩阵
images_train=images_train';
images_train = sparse(images_train);% 转化为稀疏矩阵后，速度提升很多5.41/7.79大概速度
```

```
images_test=images_test';
images_test = sparse(images_test);
% linear SVM
options='-t 2 -c 4 -g 0.015625 -q';
model=svmtrain(labels_train(1:60000), images_train(1:60000,:), options);

[y_predict,accuracy,prob_estimates]=svmpredict(labels_test, images_test, model); % 利用模型进行预测
acc = sum(y_predict==labels_test)/10000
```

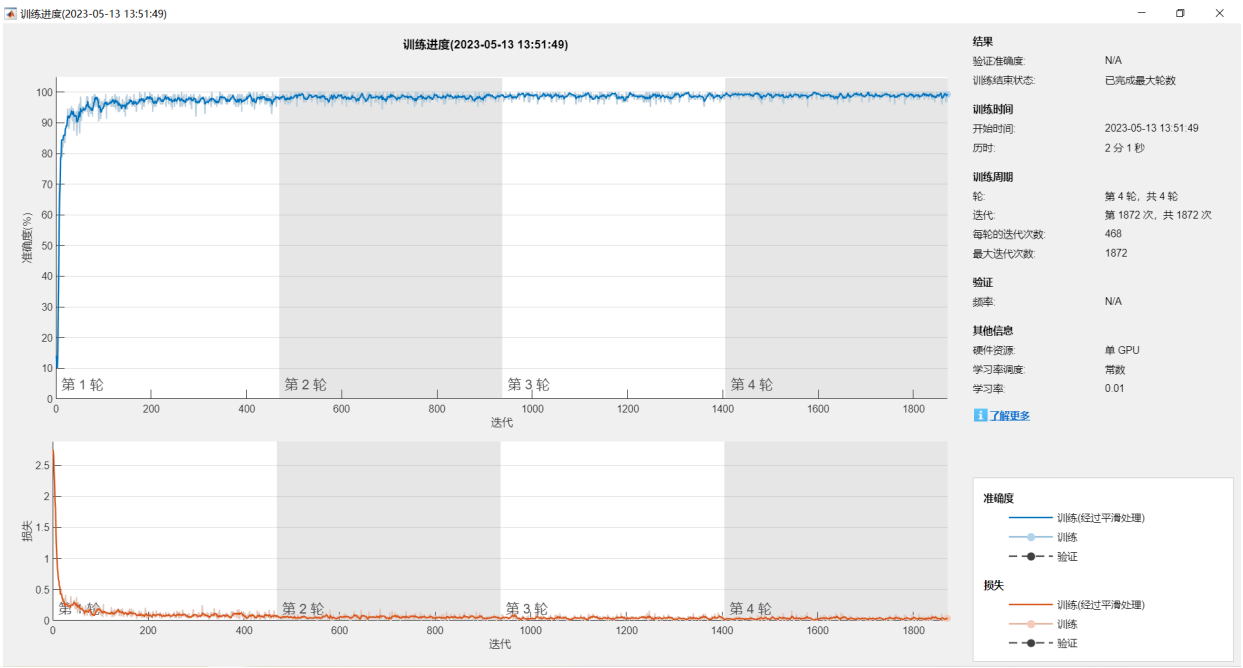
该代码的预测准确率达到98.46%，模式识别效果是目前已经测试的三种方法中最优的，运行的时长在KNN和最小二乘解之间，不算太慢。

```
[y_predict,accuracy,prob_estimates]=svmpredict(labels_test, images_test, model); % 利用模型进行预测

Accuracy = 98.46% (9846/10000) (classification)

acc = sum(y_predict==labels_test)/10000

acc = 0.9846
```

深度学习 98.75%

```
%% 读取数据
images_train=LoadMNISTImages('train-images.idx3-ubyte');
images_test=LoadMNISTImages('t10k-images.idx3-ubyte');
labels_train=LoadMNISTLabels('train-labels.idx1-ubyte');
labels_test=LoadMNISTLabels('t10k-labels.idx1-ubyte');
%% 数据格式转换
X_Train = zeros(28,28,1,60000);
%MATLAB深度学习需要构建四维矩阵，含义为长X宽X颜色通道X训练集图像数
for i = 1:60000
temp = images_train(:,i);
X_Train(:,:,1,i) = reshape(temp,28,28);
end
Y_Train = categorical(labels_train);%训练集标签的类型转换
X_Test = zeros(28,28,1,10000);%测试集数据转换
for i = 1:10000
temp = images_test(:,i);
X_Test(:,:,1,i) = reshape(temp,28,28);
end
Y_Test = categorical(labels_test);%测试集标签转换
%% 深度学习的网络结构
layers = [ ...
imageInputLayer([28,28,1]) %输入层
convolution2dLayer(3,8,'Padding','same') %3X3的8个通道卷积层， 0延拓
batchNormalizationLayer %分组归一化层
reluLayer %ReLu操作
maxPooling2dLayer(2,'Stride',2) %最大值池化层
convolution2dLayer(3,16,'Padding','same') %3X3的16个通道卷积层， 0延拓
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
```

```
convolution2dLayer(3,32,'Padding','same') %3X3的32个通道卷积层， 0延拓
batchNormalizationLayer
reluLayer
fullyConnectedLayer(10) %全连接层，对应10种分类输出
softmaxLayer %Softmax层增大输出区分度
classificationLayer %分类器层
];
%% 深度学习参数设置
options = trainingOptions('sgdm', ...
'InitialLearnRate',0.01, ...
'MaxEpochs',4, ...
'Shuffle','every-epoch', ...
'Verbose',false, ...
'Plots','training-progress');
%% 训练（一行代码）
net=trainNetwork(X_Train,Y_Train,layers, options); %训练

%% 测试
Y_pred = classify(net, X_Test); %测试
accy = sum(Y_pred == Y_Test) / length(Y_Test) %计算准确度
```



使用给出的深度学习的代码可以得到对于全部的 MNIST 数据集，识别的准确率约为98.75%，运行时间比SVC更短，兼顾了识别的准确率和速度。

```
%% 测试
Y_pred = classify(net, X_Test); %测试
accy = sum(Y_pred == Y_Test) / length(Y_Test) %计算准确度
```

```
accy = 0.9875
```

Python

先在python中读取MNIST数据集。

```python
import time
import mnist_load as mnist_load
from sklearn.metrics import classification_report  # 生产报告
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

print('reading training and testing data...')
x_train, y_train, x_test, y_test = mnist_load.get_data()
print('data: ', x_train.shape, x_test.shape)
print('label: ', y_train.shape, y_test.shape)
```

查看数据集的大小。

```
reading training and testing data...
data:  (60000, 784) (10000, 784)
label:  (60000,) (10000,)
```

使用 python 的 sklearn 包可以完成很多分类的测试，下面在 python 中对其他的方法进行尝试和检验。

随机森林 96.91%

可以通过以下代码计算出每一个分类类别的准确率以及总体的准确率，同时展示结果矩阵。（LR、SVC为方便起见仅展示结果矩阵和总体准确率）

```python
# rf
start_time = time.time()
rf = RandomForestClassifier(n_jobs=-1)
rf.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
pred_rf = rf.predict(x_test)
print('predict took %fs!' % (time.time() - start_time))
report_rf = classification_report(y_test, pred_rf)
confusion_mat_rf = confusion_matrix(y_test, pred_rf)
print(report_rf)
print(confusion_mat_rf)
print('随机森林准确率: %0.4lf' % accuracy_score(pred_rf, y_test))
```

下图可以看出，使用随机森林的算法，每一个分类类别的识别准确率都达到了95%或以上，其中3、8、9的识别准确率较低。

```
training took 11.032089s!
predict took 0.131099s!
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.96      1032
           3       0.95      0.96      0.96      1010
           4       0.97      0.97      0.97       982
           5       0.98      0.96      0.97       892
           6       0.97      0.97      0.97       958
           7       0.97      0.97      0.97      1028
           8       0.96      0.95      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

下图展示随机森林算法的结果矩阵，主要集中分布在主对角线上，说明绝大多数的识别结果是正确的，同时也可以看出手写的数字除了对应的分类外更加倾向于“被算法认为是”的分类。如数字3容易和5、9混杂等。但是总体的准确率依然达到了96.91%，识别的效果较好。

```
[[ 970    0    1    0    0    1    4    1    3    0]
 [   0 1122    3    4    0    1    3    1    1    0]
 [   6    0  997    7    2    0    4    8    7    1]
 [   0    0    9  972    0    7    0    9   10    3]
 [   1    0    0    0  956    0    5    0    3   17]
 [   2    1    1   14    1  859    5    1    4    4]
 [   6    3    1    1    5    5  934    0    3    0]
 [   1    2   18    3    0    0    0  993    2    9]
 [   5    0    4    8    6    5    4    5  929    8]
 [   9    5    1    9   13    1    1    4    7  959]]
随机森林准确率: 0.9691
```

LR 92.55%

```
# LR
start_time = time.time()
lr = LogisticRegression()
lr.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
pred_lr = lr.predict(x_test)
print('predict took %fs!' % (time.time() - start_time))
report_lr = classification_report(y_test, pred_lr)
confusion_mat_lr = confusion_matrix(y_test, pred_lr)
print(report_lr)
print(confusion_mat_lr)
print('LR准确率: %0.4lf' % accuracy_score(pred_lr, y_test))
```

可以看出LR算法分类的准确率约为92.55%，略低于随机森林的算法。数字5、8出现的分类错误较多。

```
[[ 963    0    0    3    1    3    4    4    2    0]
 [   0 1112    4    2    0    1    3    2   11    0]
 [   3   10  926   15    6    4   15    8   42    3]
 [   4    1   21  916    1   26    3    9   22    7]
 [   1    1    7    3  910    0    9    7   10   34]
 [  11    2    1   33   11  776   11    6   35    6]
 [   9    3    7    3    7   16  910    2    1    0]
 [   1    6   24    5    7    1    0  951    3   30]
 [   8    7    6   23    6   26   10   10  869    9]
 [   9    7    0   11   25    6    0   22    7  922]]
LR准确率: 0.9255
```

SVC 97.92%

采用支持向量机辅助分类的方法也可以通过 sklearn 包中函数实现。

```
# SVM
svm = SVC()
svm.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
pred_svm = svm.predict(x_test)
report_svm = classification_report(y_test, pred_svm)
print('predict took %fs!' % (time.time() - start_time))
confusion_mat_svm = confusion_matrix(y_test, pred_svm)
print(report_svm)
print(confusion_mat_svm)
print('SVC准确率: %0.4lf' % accuracy_score(pred_svm, y_test))
```

可以看出SVC算法分类的准确率约为97.92%，分类的准确性最好，耗时明显比其他方法更久。

```
[[ 973    0    1    0    0    2    1    1    2    0]
 [   0 1126    3    1    0    1    1    1    2    0]
 [   6    1 1006    2    1    0    2    7    6    1]
 [   0    0    2  995    0    2    0    5    5    1]
 [   0    0    5    0  961    0    3    0    2   11]
 [   2    0    0    9    0  871    4    1    4    1]
 [   6    2    0    0    2    3  944    0    1    0]
 [   0    6   11    1    1    0    0  996    2   11]
 [   3    0    2    6    3    2    2    3  950    3]
 [   3    4    1    7   10    2    1    7    4  970]]
SVC准确率: 0.9792
```

比较三种方法的耗时

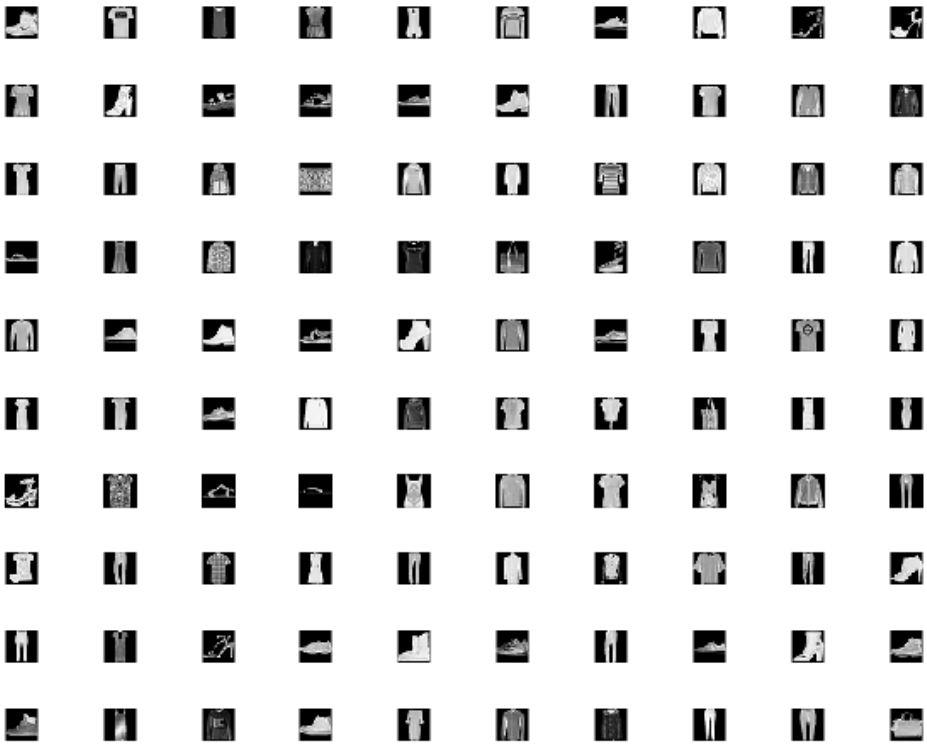
- 随机森林：训练耗时 11.032089s，预测耗时 0.131099s
- LR：训练耗时 16.361046s，预测耗时 0.033863s
- SVC：训练耗时 225.194740s，预测耗时 70.815080s

推广到Fashion-MNIST数据集

此外，可以尝试将课件的方法推广到Fashion-MNIST或cifar-10数据集（非必做），用以证明自身方法是否有效。

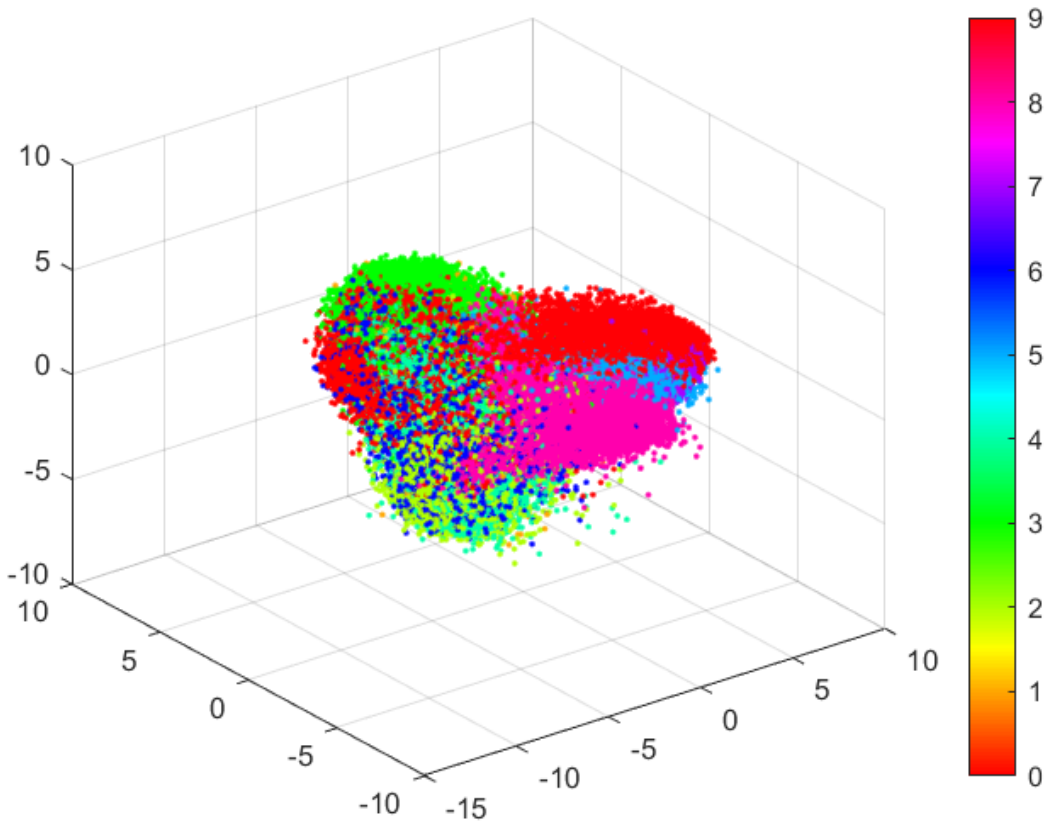
Matlab

先查看数据集中给出的图像。



用主成分分析和降维方法图示训练集 84.85%

```
images = LoadMNISTImages('train-images-idx3-ubyte');
labels = LoadMNISTLabels('train-labels-idx1-ubyte');
images_centered = images - mean(images,2);
PCA_M = images_centered*images_centered';
[V,D] = eigs(PCA_M,784,'la');
images_coordinate = V'*images_centered;
images_PC = images_coordinate(1:3,:);
scatter3(images_PC(1,:),images_PC(2,:),images_PC(3,:),[],labels,'.');
colormap hsv,colorbar
```




```
images_train=LoadMNISTImages('train-images-idx3-ubyte');
images_test=LoadMNISTImages('t10k-images-idx3-ubyte');
labels_train=LoadMNISTLabels('train-labels-idx1-ubyte');
labels_test=LoadMNISTLabels('t10k-labels-idx1-ubyte');
test_image = images_test(:,20);
temp1 = reshape(images_test(:,20),28,28);
imshow(temp1,[])
x_diff = images_train - test_image;
x_dist = sqrt(sum(x_diff.*x_diff));
[~,x_ranking] = sort(x_dist);
KNN_labels = labels_train(x_ranking(1:30));
KNN_labels'
GroundTruth_label = labels_test(20)
```

```
test_image = images_test(:,20);
temp1 = reshape(images_test(:,20),28,28);
imshow(temp1,[])
```

```
x_diff = images_train - test_image;
%计算训练集和该图像的差距
x_dist = sqrt(sum(x_diff.*x_diff));
%根据差距计算出欧氏距离
[~,x_ranking] = sort(x_dist);
%排序，无需返回排序后的序列，只返回排序后从小到大的列指标
KNN_labels = labels_train(x_ranking(1:30));
KNN_labels'
```

```
ans = 1×30
     0     0     0     6     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     6     0     0     0     0     6     0     0
```

```
GroundTruth_label = labels_test(20)
```

```
GroundTruth_label = 0
```

可以看出使用 KNN 方法判断出和原图像“距离最近”的30幅图大多数标签都与第20幅图一致，标签为0，少部分显示标签为6，下面查看出现错误的几个图像。

```
temp2 = reshape(images_train(:,x_ranking(4)),28,28);
imshow(temp2,[])
```

```
temp3 = reshape(images_train(:,x_ranking(23)),28,28);
imshow(temp3,[])
```

```
temp4 = reshape(images_train(:,x_ranking(28)),28,28);
imshow(temp4,[])
```

发现图像和示例图像很相似，大概是连衣裙和宽松短袖上衣的区别。

对 KNN 模式识别的方法进行评分并计算其正确率。

```
Correct_Count = 0;
for j = 1:10000
x_diff = images_train - images_test(:,j);
x_dist = sqrt(sum(x_diff.*x_diff));
[~,x_ranking] = sort(x_dist);
KNN_labels = labels_train(x_ranking(1:20));
temp = zeros(1,10);
for i = 1:20
temp(KNN_labels(i)+1) = temp(KNN_labels(i)+1) + (21-i);
end
[~,Final_label] = max(temp);
Final_label = Final_label - 1;
GroundTruth_label = labels_test(j);
if(Final_label == GroundTruth_label)
Correct_Count = Correct_Count + 1;
end
end
```

```
Correct_rate = Correct_Count/10000
```

```
Correct_rate = Correct_Count/10000

Correct_rate = 0.8485
```

得到该方法的正确率大约为84.85%，比使用MNIST数据集进行的测试正确率要低得多，经过对标签和图像的检查，产生的识别错误可能是因为上面示例中提及的难以用距离区分宽松短袖上衣和连衣裙等原因。

最小二乘解 80.87%

直接对训练集进行线性拟合，计算其最小二乘解，使得解尽可能满足方程组。

```
train_amount = 60000;%假设数据已经读取完毕，全部60000张训练集用于训练
B = zeros(10,train_amount);
for i = 1:train_amount
B(labels_train(i)+1,i)=1;%设置矩阵B的值
end
AT = images_train(:,1:train_amount)';%将训练集转置
X = (AT\B')';%MATLAB命令可直接获得最小二乘解
%% Test Data
Output_new = X*images_test;%乘以测试集后会得到10000组概率向量
[~,label_new] = max(Output_new);%获得每一组向量的最大下标所在位置
label_new = label_new -1; %位置-1即为预估的手写数字值
accu = labels_test==label_new';
sum(accu)/10000 %准确率的计算
```

该方法推广到 Fashion-MNIST 数据集中同样是运行速度最快且准确率最低的，相比起在 MNIST 中的实践准确率会更低一些，约为80.87%，只能应用在需要短时间内进行粗糙分类的情况。

```
sum(accu)/10000 %准确率的计算

ans = 0.8087
```

SVC 90.15%

```
images_train=LoadMNISTImages('train-images-idx3-ubyte');
images_test=LoadMNISTImages('t10k-images-idx3-ubyte');
labels_train=LoadMNISTLabels('train-labels-idx1-ubyte');
labels_test=LoadMNISTLabels('t10k-labels-idx1-ubyte');

% transpose 转置后，每一行为一幅图像 变为一个number*pixel的矩阵
images_train=images_train';
images_train = sparse(images_train);
images_test=images_test';
images_test = sparse(images_test);
% linear SVM
options='-t 2 -c 4 -g 0.015625 -q';
model=svmtrain(labels_train(1:60000), images_train(1:60000,:), options); % 训练数据得到模型
```

```
14
15  [y_predict,accuracy,prob_estimates]=svmpredict(labels_test, images_test,
    model); % 利用模型进行预测
16  acc = sum(y_predict==labels_test)/10000
```

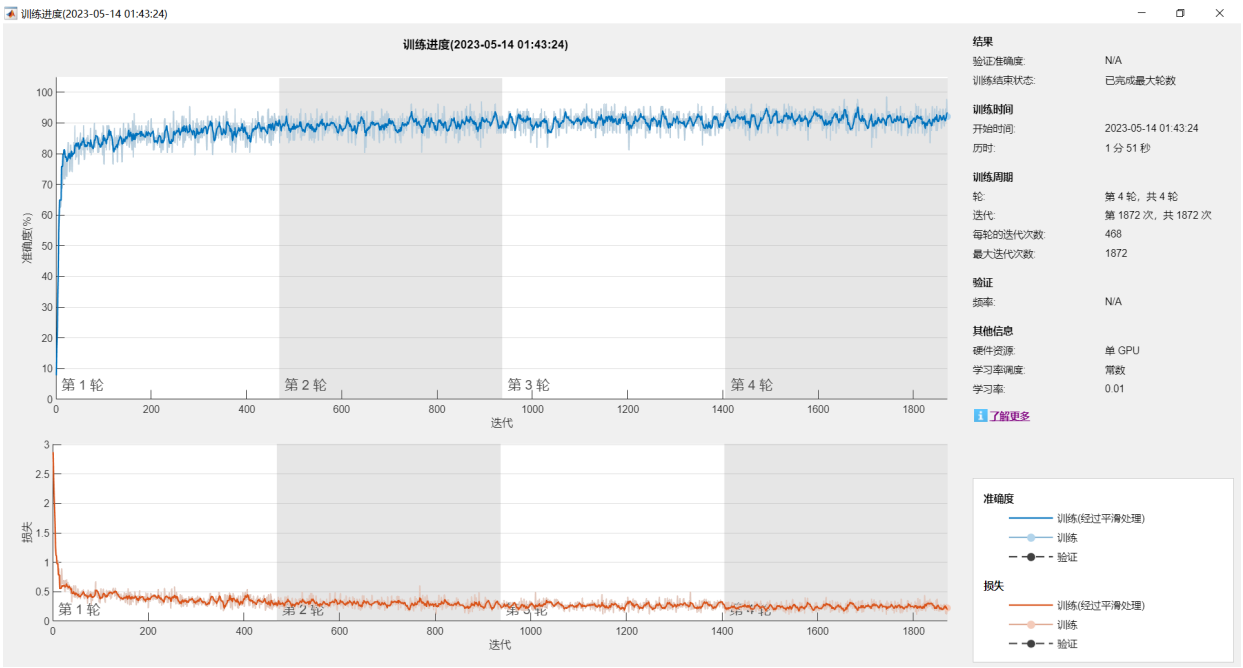
该代码的预测准确率约为90.15%，模式识别效果是目前已经测试的三种方法中最优的，运行的时长在KNN和最小二乘解之间。

```
[y_predict,accuracy,prob_estimates]=svmpredict(labels_test, images_test, model); % 利用模型进行预测

Accuracy = 90.15% (9015/10000) (classification)

acc = sum(y_predict==labels_test)/10000

acc = 0.9015
```

深度学习 90.19%

```
1   %% 读取数据
2   images_train=LoadMNISTImages('train-images-idx3-ubyte');
3   images_test=LoadMNISTImages('t10k-images-idx3-ubyte');
4   labels_train=LoadMNISTLabels('train-labels-idx1-ubyte');
5   labels_test=LoadMNISTLabels('t10k-labels-idx1-ubyte');
6   %% 数据格式转换
7   X_Train = zeros(28,28,1,60000);
8   %MATLAB深度学习需要构建四维矩阵，含义为长X宽X颜色通道X训练集图像数
9   for i = 1:60000
10  temp = images_train(:,i);
11  X_Train(:,:,1,i) = reshape(temp,28,28);
12  end
13  Y_Train = categorical(labels_train);%训练集标签的类型转换
14  X_Test = zeros(28,28,1,10000);%测试集数据转换
15  for i = 1:10000
16  temp = images_test(:,i);
17  X_Test(:,:,1,i) = reshape(temp,28,28);
18  end
19  Y_Test = categorical(labels_test);%测试集标签转换
20  %% 深度学习的网络结构
21  layers = [ ...
22  imageInputLayer([28,28,1]) %输入层
23  convolution2dLayer(3,8,'Padding','same') %3X3的8个通道卷积层， 0延拓
24  batchNormalizationLayer %分组归一化层
25  reluLayer %ReLu操作
26  maxPooling2dLayer(2,'Stride',2) %最大值池化层
27  convolution2dLayer(3,16,'Padding','same') %3X3的16个通道卷积层， 0延拓
28  batchNormalizationLayer
29  reluLayer
30  maxPooling2dLayer(2,'Stride',2)
31  convolution2dLayer(3,32,'Padding','same') %3X3的32个通道卷积层， 0延拓
32  batchNormalizationLayer
33  reluLayer
34  fullyConnectedLayer(10) %全连接层，对应10种分类输出
35  softmaxLayer %Softmax层增大输出区分度
```

```
classificationLayer %分类器层
];
%% 深度学习参数设置
options = trainingOptions('sgdm', ...
'InitialLearnRate',0.01, ...
'MaxEpochs',4, ...
'Shuffle','every-epoch', ...
'Verbose',false, ...
'Plots','training-progress');
%% 训练（一行代码）
net=trainNetwork(X_Train,Y_Train,layers, options); %训练

%% 测试
Y_pred = classify(net, X_Test); %测试
accy = sum(Y_pred == Y_Test) / length(Y_Test) %计算准确度
```



使用给出的深度学习的代码可以得到对于全部的 Fashion-MNIST 数据集，识别的准确率约为90.19%，在准确率和SVC基本一致的情况下运行时间比SVC更短，兼顾了识别的准确率和速度。

```
%% 测试
Y_pred = classify(net, X_Test); %测试
accy = sum(Y_pred == Y_Test) / length(Y_Test) %计算准确度
```

```
accy = 0.9019
```

Python

在 python 内置的深度学习包中下载并分割 MNIST 数据集，查看数据集的大小。

```
import time
from sklearn.metrics import classification_report
import fashion_mnist_load as mnist_load
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

print('reading training and testing data...')
x_train, y_train, x_test, y_test = mnist_load.get_data()
```

下面尝试将随机森林、LR、SVM的方法推广到 Fashion-MNIST 中。

随机森林 87.78%

可以通过以下代码计算出每一个分类类别的准确率以及总体的准确率，同时展示结果矩阵。（LR、SVC为方便起见仅展示结果矩阵和总体准确率）

```
# rf
start_time = time.time()
rf = RandomForestClassifier(n_jobs=-1)
rf.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
# 根据模型做预测，返回预测结果
pred_rf = rf.predict(x_test)
print('predict took %fs!' % (time.time() - start_time))
report_rf = classification_report(y_test, pred_rf)
confusion_mat_rf = confusion_matrix(y_test, pred_rf)
print(report_rf)
print(confusion_mat_rf)
print('随机森林准确率: %0.4lf' % accuracy_score(pred_rf, y_test))
```

下图可以看出，在 Fashion-MNIST 数据集中使用随机森林的算法准确度明显不及 MNIST 的手写识别，可能是由于服饰之间的形状区别比手写数字之间的区别更加精细的原因。部分分类类别的识别准确率不及80%，类别2、4、6的识别准确率较低。

```
reading training and testing data...
training took 21.331156s!
predict took 0.143479s!
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      1000
           1       0.99      0.96      0.98      1000
           2       0.76      0.80      0.78      1000
           3       0.88      0.91      0.89      1000
           4       0.76      0.81      0.79      1000
           5       0.98      0.96      0.97      1000
           6       0.72      0.58      0.65      1000
           7       0.93      0.95      0.94      1000
           8       0.96      0.97      0.97      1000
           9       0.95      0.95      0.95      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

下图展示随机森林算法的结果矩阵，主要集中分布在主对角线上，说明绝大多数的识别结果是正确的，同时也可以看出部分服饰类别难以和其他的类别区分，如类别6，和在 Matlab 使用主成分分析法时出现的情况吻合。总体的准确率大约为87.78%，整体识别的效果还不错。

```
[[871   0  11  25   2   1  79   0  11   0]
 [  2 964   3  20   4   0   6   0   1   0]
 [ 15   0 798  11 118   0  55   0   3   0]
 [ 19   2  13 908  28   0  28   0   2   0]
 [  0   0  99  39 812   0  48   0   2   0]
 [  0   0   0   1   0 962   0  26   1  10]
 [158   1 118  27  95   0 585   0  16   0]
 [  0   0   0   0   0  11   0 954   0  35]
 [  0   2   5   2   5   2   8   4 971   1]
 [  0   0   0   0   0   7   1  37   2 953]]
随机森林准确率: 0.8778
```

LR 84.12%

```
# LR
start_time = time.time()
lr = LogisticRegression()
lr.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
# 根据模型做预测，返回预测结果
pred_lr = lr.predict(x_test)
print('predict took %fs!' % (time.time() - start_time))
report_lr = classification_report(y_test, pred_lr)
confusion_mat_lr = confusion_matrix(y_test, pred_lr)
print(report_lr)
print(confusion_mat_lr)
print('LR准确率: %0.4lf' % accuracy_score(pred_lr, y_test))
```

可以看出LR算法分类的准确率约为84.12%，明显低于随机森林的算法。和在 MNIST 数据集中出现的情况一致。

```
[[812   5  16  46   9   0  98   0  14   0]
 [  2 960   1  27   4   0   4   0   2   0]
 [ 18   6 737  11 140   1  78   0   9   0]
 [ 25  15  15 858  44   1  37   0   5   0]
 [  0   3 106  33 779   1  70   0   8   0]
 [  1   1   0   0   0 891   0  56   9  42]
 [139   3 129  42 114   0 550   0  23   0]
 [  0   0   0   0   0  36   0 933   0  31]
 [  3   1   7  10   2   3  21   5 947   1]
 [  0   0   0   0   0  13   0  39   3 945]]
LR准确率: 0.8412
```

SVC 88.28%

```
# SVM
svm = SVC()
svm.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))
start_time = time.time()
# 根据模型做预测，返回预测结果
pred_svm = svm.predict(x_test)
report_svm = classification_report(y_test, pred_svm)
print('predict took %fs!' % (time.time() - start_time))
confusion_mat_svm = confusion_matrix(y_test, pred_svm)
print(report_svm)
print(confusion_mat_svm)
print('SVC准确率: %0.4lf' % accuracy_score(pred_svm, y_test))
```

可以看出SVC算法分类的准确率约为88.28%，分类的准确性最好，耗时明显比其他方法更久。

```
[[857   0  16  28   3   2  85   0   9   0]
 [  4 962   2  25   3   0   4   0   0   0]
 [ 11   2 816  16  88   0  65   0   2   0]
 [ 27   3  11 890  33   0  32   0   4   0]
 [  1   1  87  32 815   0  61   0   3   0]
 [  0   0   0   1   0 951   0  33   1  14]
 [135   1 104  27  68   0 654   0  11   0]
 [  0   0   0   0   0  21   0 955   0  24]
 [  3   1   1   5   2   2   4   5 977   0]
 [  0   0   0   0   0  11   1  37   0 951]]
SVC准确率: 0.8828
```

比较三种方法的耗时

- 随机森林：训练耗时 21.331156s，预测耗时 0.143479s
- LR：训练耗时 18.474124s，预测耗时 0.046353s
- SVC：训练耗时 241.407014s，预测耗时 106.894543s

Tensorflow搭建神经网络 89.73%

```
import tensorflow as tf
# Load the dataset
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.fashion_mnist.load_data()

```

```
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
  tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
  tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(units=128, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(units=10, activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
Epoch 1/5
1875/1875 [==============================] - 69s 36ms/step - loss: 0.5613 - accuracy: 0.7987 - val_loss: 0.3832 - val_accuracy: 0.8585
Epoch 2/5
1875/1875 [==============================] - 69s 37ms/step - loss: 0.3781 - accuracy: 0.8630 - val_loss: 0.3225 - val_accuracy: 0.8812
Epoch 3/5
1875/1875 [==============================] - 67s 36ms/step - loss: 0.3272 - accuracy: 0.8814 - val_loss: 0.2963 - val_accuracy: 0.8924
Epoch 4/5
1875/1875 [==============================] - 66s 35ms/step - loss: 0.2970 - accuracy: 0.8915 - val_loss: 0.2803 - val_accuracy: 0.8943
Epoch 5/5
1875/1875 [==============================] - 66s 35ms/step - loss: 0.2730 - accuracy: 0.9004 - val_loss: 0.2801 - val_accuracy: 0.8973
313/313 [==============================] - 3s 9ms/step - loss: 0.2801 - accuracy: 0.8973
Test accuracy: 0.8973000049591064
```