Q4

May 12, 2024

0.1 Mahdi Anvari 610700002 Homework 2 of Machine Learning Question 4

[2]:
```
# importing libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sbn
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
```

```
WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\losses.py:2664: The name tf.losses.sparse_softmax_cross_entropy
is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.
```

- Load MNIST dataset (could be accessed using from keras.datasets in python)

[12]:
```
mnist_path = 'c:/Users/M/Downloads/mnist.npz'
with np.load(mnist_path, allow_pickle=True) as f:
    X_train, Y_train = f['x_train'], f['y_train']
    X_test, Y_test = f['x_test'], f['y_test']
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

[13]:
```
X_train = X_train.reshape(X_train.shape[0],-1)
X_test = X_test.reshape(X_test.shape[0],-1)
```

```
print(X_train.shape)
print(X_test.shape)

scaler = StandardScaler()
NormalizedXtrain = scaler.fit_transform(X_train)
NormalizedXtest = scaler.fit_transform(X_test)
```

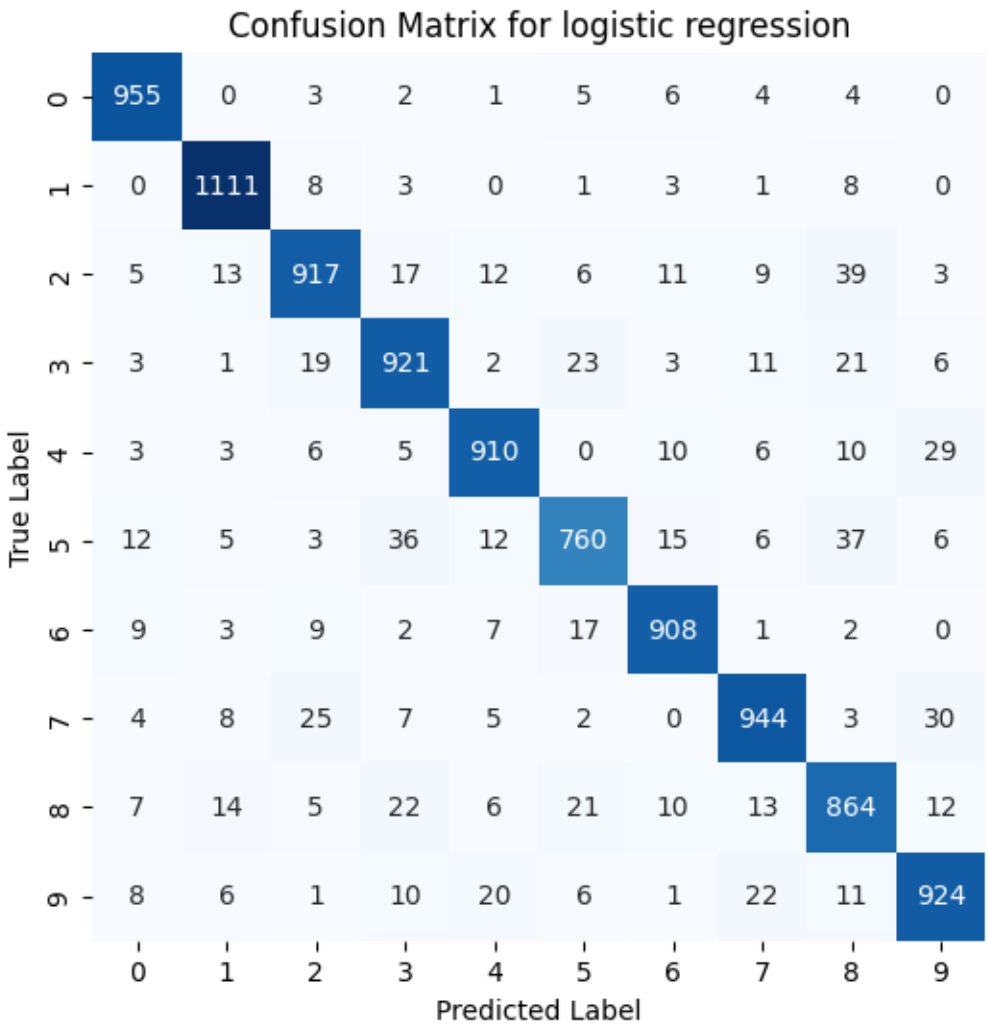
```
(60000, 784)
(10000, 784)
```

a. Logistic Regression

[23]:
```
LogReg = LogisticRegression(max_iter=1000)
LogReg.fit(X_train, Y_train)
Y_pred = LogReg.predict(X_test)
LogAccuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy for logistic regression:", LogAccuracy)
```

```
Accuracy for logistic regression: 0.9214

C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[29]:
```
LogCM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(LogCM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for logistic regression')
plt.show()
```



b. MLP with one hidden layer of size 128

[52]:
```
modelB = Sequential(name='modelB')
modelB.add(Dense(units=128, activation='relu', input_shape=(784,)␣
 ↪,name='hidden_layer'))
modelB.add(Dense(units=10, activation='softmax', name='output_layer'))
modelB.compile(optimizer=tf.optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
modelB.summary()
```

```
Model: "modelB"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 hidden_layer (Dense)         (None, 128)               100480
```

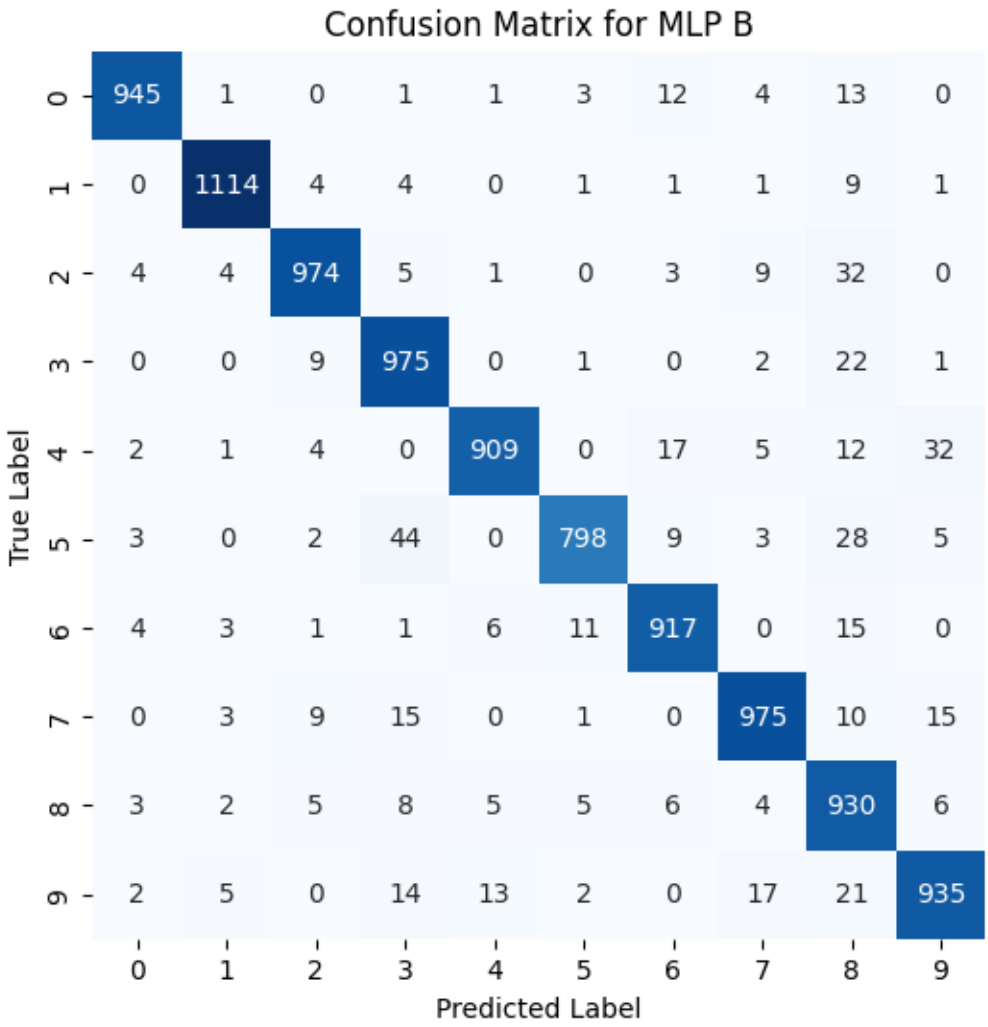
```
 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[55]:
```
modelB.fit(X_train, Y_train, epochs=10, batch_size=32, validation_split=0.2)
loss, accuracy = modelB.evaluate(X_test, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1760 -
accuracy: 0.9571 - val_loss: 0.2726 - val_accuracy: 0.9454
Epoch 2/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1693 -
accuracy: 0.9594 - val_loss: 0.3052 - val_accuracy: 0.9431
Epoch 3/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1688 -
accuracy: 0.9591 - val_loss: 0.2883 - val_accuracy: 0.9471
Epoch 4/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1683 -
accuracy: 0.9614 - val_loss: 0.2503 - val_accuracy: 0.9507
Epoch 5/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.1550 -
accuracy: 0.9623 - val_loss: 0.2626 - val_accuracy: 0.9504
Epoch 6/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1514 -
accuracy: 0.9624 - val_loss: 0.3094 - val_accuracy: 0.9448
Epoch 7/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1561 -
accuracy: 0.9629 - val_loss: 0.3498 - val_accuracy: 0.9437
Epoch 8/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1581 -
accuracy: 0.9639 - val_loss: 0.3044 - val_accuracy: 0.9467
Epoch 9/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1450 -
accuracy: 0.9649 - val_loss: 0.3202 - val_accuracy: 0.9392
Epoch 10/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1398 -
accuracy: 0.9662 - val_loss: 0.3565 - val_accuracy: 0.9493
313/313 [==============================] - 1s 2ms/step - loss: 0.3818 -
accuracy: 0.9472
Test Loss: 0.3817773759365082
Test Accuracy: 0.9472000002861023
```

[56]:
```
Y_pred = np.argmax(modelB.predict(X_test), axis=-1)
B_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(B_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for MLP B')
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```



c. MLP with two hidden layers of sizes 256 and 128.

[57]:
```
modelC = Sequential(name='modelC')
modelC.add(Dense(units=256, activation='relu', input_shape=(784,)␣
↪,name='hidden_layer1'))
```

```
modelC.add(Dense(units=128, activation='relu' ,name='hidden_layer2'))
modelC.add(Dense(units=10, activation='softmax', name='output_layer'))
modelC.compile(optimizer=tf.optimizers.Adam(),␣
  ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
modelC.summary()
```

```
Model: "modelC"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden_layer1 (Dense)       (None, 256)               200960

 hidden_layer2 (Dense)       (None, 128)               32896

 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

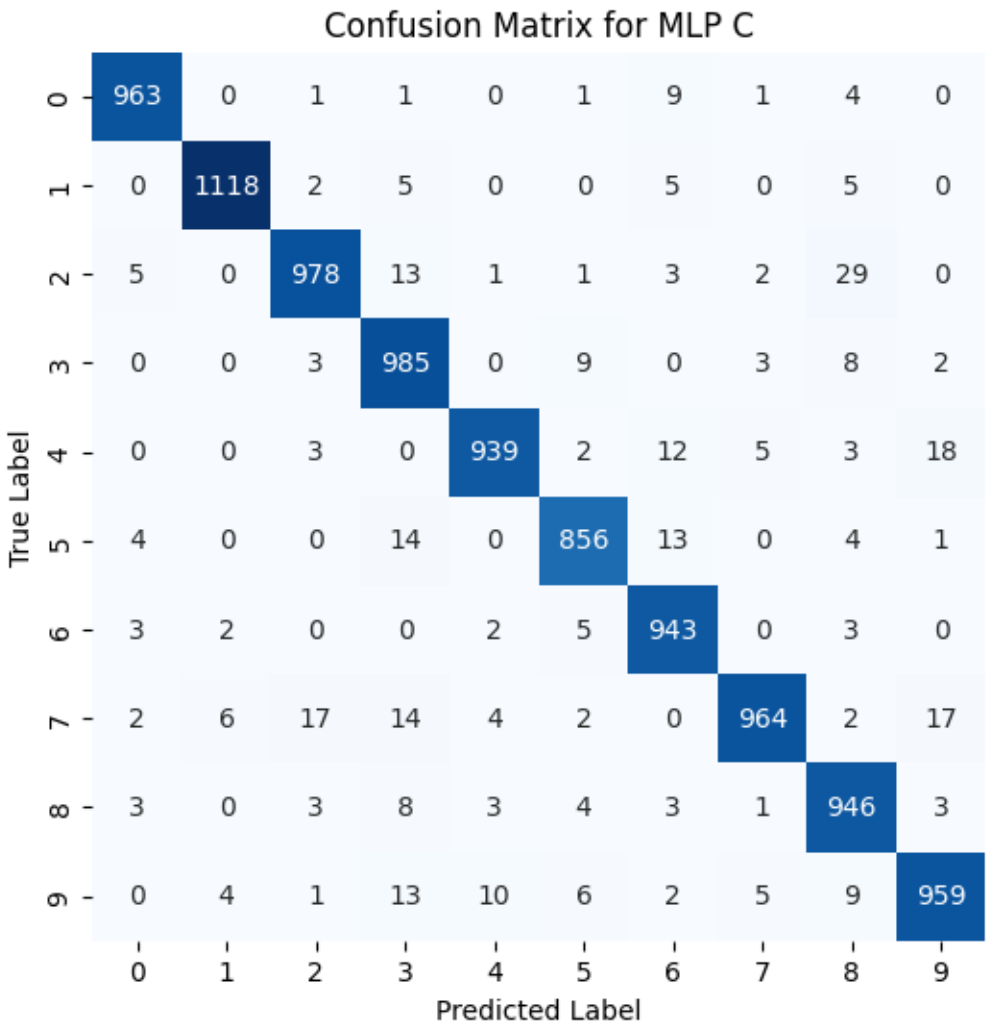
[58]:
```
modelC.fit(X_train, Y_train, epochs=10, batch_size=32, validation_split=0.2)
loss, accuracy = modelC.evaluate(X_test, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
1500/1500 [==============================] - 8s 5ms/step - loss: 1.9098 -
accuracy: 0.8776 - val_loss: 0.3792 - val_accuracy: 0.9073
Epoch 2/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2613 -
accuracy: 0.9338 - val_loss: 0.2744 - val_accuracy: 0.9303
Epoch 3/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1915 -
accuracy: 0.9483 - val_loss: 0.2091 - val_accuracy: 0.9453
Epoch 4/10
1500/1500 [==============================] - 8s 6ms/step - loss: 0.1589 -
accuracy: 0.9565 - val_loss: 0.1656 - val_accuracy: 0.9578
Epoch 5/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1468 -
accuracy: 0.9595 - val_loss: 0.1998 - val_accuracy: 0.9540
Epoch 6/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1249 -
accuracy: 0.9657 - val_loss: 0.1580 - val_accuracy: 0.9590
Epoch 7/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1206 -
accuracy: 0.9674 - val_loss: 0.1565 - val_accuracy: 0.9606
```

```
Epoch 8/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1074 -
accuracy: 0.9711 - val_loss: 0.1549 - val_accuracy: 0.9647
Epoch 9/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0937 -
accuracy: 0.9754 - val_loss: 0.1802 - val_accuracy: 0.9617
Epoch 10/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0929 -
accuracy: 0.9766 - val_loss: 0.1472 - val_accuracy: 0.9653
313/313 [==============================] - 1s 3ms/step - loss: 0.1425 -
accuracy: 0.9651
Test Loss: 0.14246425032615662
Test Accuracy: 0.9650999903678894
```

[59]:
```
Y_pred = np.argmax(modelC.predict(X_test), axis=-1)
C_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(C_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for MLP C')
plt.show()
```

```
313/313 [==============================] - 1s 3ms/step
```



d. CNN with two "convolution + max pooling" blocks and a dense network with one hidden layer of size 128.

[11]:
```
modelD = Sequential(name='modelD')
modelD.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu',␣
 ↪input_shape=(28,28,1), name='CNN1'))
modelD.add(MaxPooling2D(pool_size=(2,2), name='Pooling1'))
modelD.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu',␣
 ↪name='CNN2'))
modelD.add(MaxPooling2D(pool_size=(2,2), name='Pooling2'))
modelD.add(Flatten())
modelD.add(Dense(units=128, activation='relu' ,name='hidden_layer'))
modelD.add(Dense(units=10, activation='softmax', name='output_layer'))
modelD.compile(optimizer=tf.optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
modelD.summary()
```

```
Model: "modelD"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 CNN1 (Conv2D)               (None, 26, 26, 32)        320

 Pooling1 (MaxPooling2D)     (None, 13, 13, 32)        0

 CNN2 (Conv2D)               (None, 11, 11, 64)        18496

 Pooling2 (MaxPooling2D)     (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 hidden_layer (Dense)        (None, 128)               204928

 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

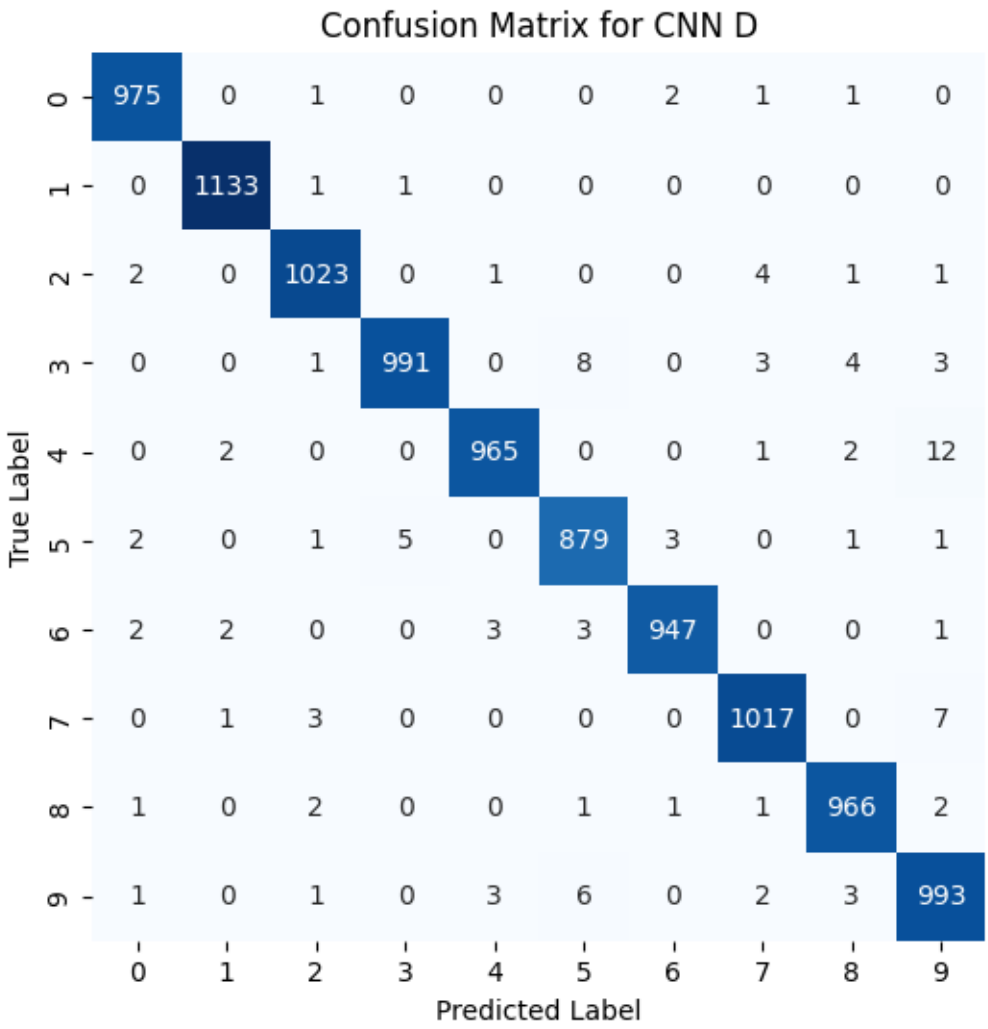
[15]:
```
ReshapedNormalizedXtrain = NormalizedXtrain.reshape(NormalizedXtrain.
 ↪shape[0],28,28,1)
ReshapedNormalizedXtest = NormalizedXtest.reshape(NormalizedXtest.
 ↪shape[0],28,28,1)

modelD.fit(ReshapedNormalizedXtrain, Y_train, epochs=10, batch_size=32,␣
 ↪validation_split=0.2)
loss, accuracy = modelD.evaluate(ReshapedNormalizedXtest, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\utils\tf_utils.py:490: The name tf.ragged.RaggedTensorValue is
deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\engine\base_layer_utils.py:380: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.
```

```
1500/1500 [==============================] - 20s 12ms/step - loss: 0.1380 -
accuracy: 0.9576 - val_loss: 0.0637 - val_accuracy: 0.9817
Epoch 2/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0452 -
accuracy: 0.9852 - val_loss: 0.0451 - val_accuracy: 0.9879
Epoch 3/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0334 -
accuracy: 0.9894 - val_loss: 0.0396 - val_accuracy: 0.9887
Epoch 4/10
1500/1500 [==============================] - 16s 10ms/step - loss: 0.0220 -
accuracy: 0.9926 - val_loss: 0.0430 - val_accuracy: 0.9873
Epoch 5/10
1500/1500 [==============================] - 16s 11ms/step - loss: 0.0171 -
accuracy: 0.9944 - val_loss: 0.0439 - val_accuracy: 0.9888
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0164 -
accuracy: 0.9947 - val_loss: 0.0483 - val_accuracy: 0.9877
Epoch 7/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0121 -
accuracy: 0.9963 - val_loss: 0.0579 - val_accuracy: 0.9868
Epoch 8/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0091 -
accuracy: 0.9969 - val_loss: 0.0467 - val_accuracy: 0.9897
Epoch 9/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0078 -
accuracy: 0.9974 - val_loss: 0.0527 - val_accuracy: 0.9893
Epoch 10/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0078 -
accuracy: 0.9976 - val_loss: 0.0599 - val_accuracy: 0.9894
313/313 [==============================] - 2s 6ms/step - loss: 0.0506 -
accuracy: 0.9889
Test Loss: 0.050574351102113724
Test Accuracy: 0.9889000058174133
```

[16]:
```
Y_pred = np.argmax(modelD.predict(ReshapedNormalizedXtest), axis=-1)
D_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(D_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for CNN D')
plt.show()
```

```
313/313 [==============================] - 2s 5ms/step
```


Confusion Matrix for CNN D
True Label
Predicted Label
0 975 0 1 0 0 0 2 1 1 0
1 0 1133 1 1 0 0 0 0 0 0
2 2 0 1023 0 1 0 0 4 1 1
3 0 0 1 991 0 8 0 3 4 3
4 0 2 0 0 965 0 0 1 2 12
5 2 0 1 5 0 879 3 0 1 1
6 2 2 0 0 3 3 947 0 0 1
7 0 1 3 0 0 0 0 1017 0 7
8 1 0 2 0 0 1 1 1 966 2
9 1 0 1 0 3 6 0 2 3 993
0 1 2 3 4 5 6 7 8 9

[ ]: