In [ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-d
atasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-data
sets/mnist.npz)
11490434/11490434 [==============================] - 0s 0us/step
```

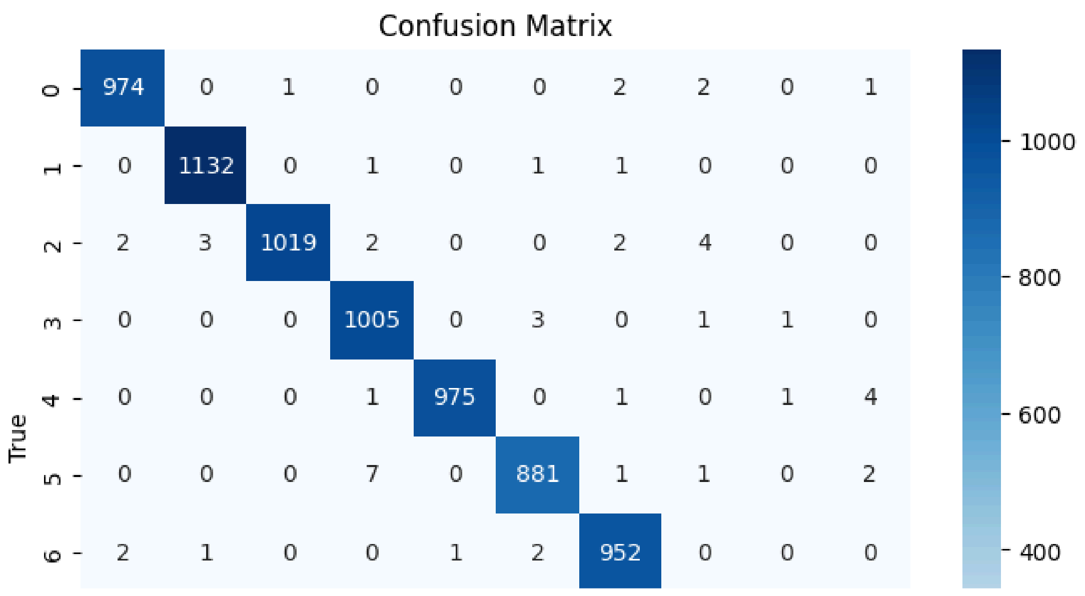
In [ ]:
```
# Preprocess the data
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
```

In [ ]:
```
# Define the CNN architecture
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model
```

In [ ]:
```
# Define K-Fold cross-validation
kfold = KFold(n_splits=5, shuffle=True)

fold = 0
accuracies = []
conf_matrices = []
```

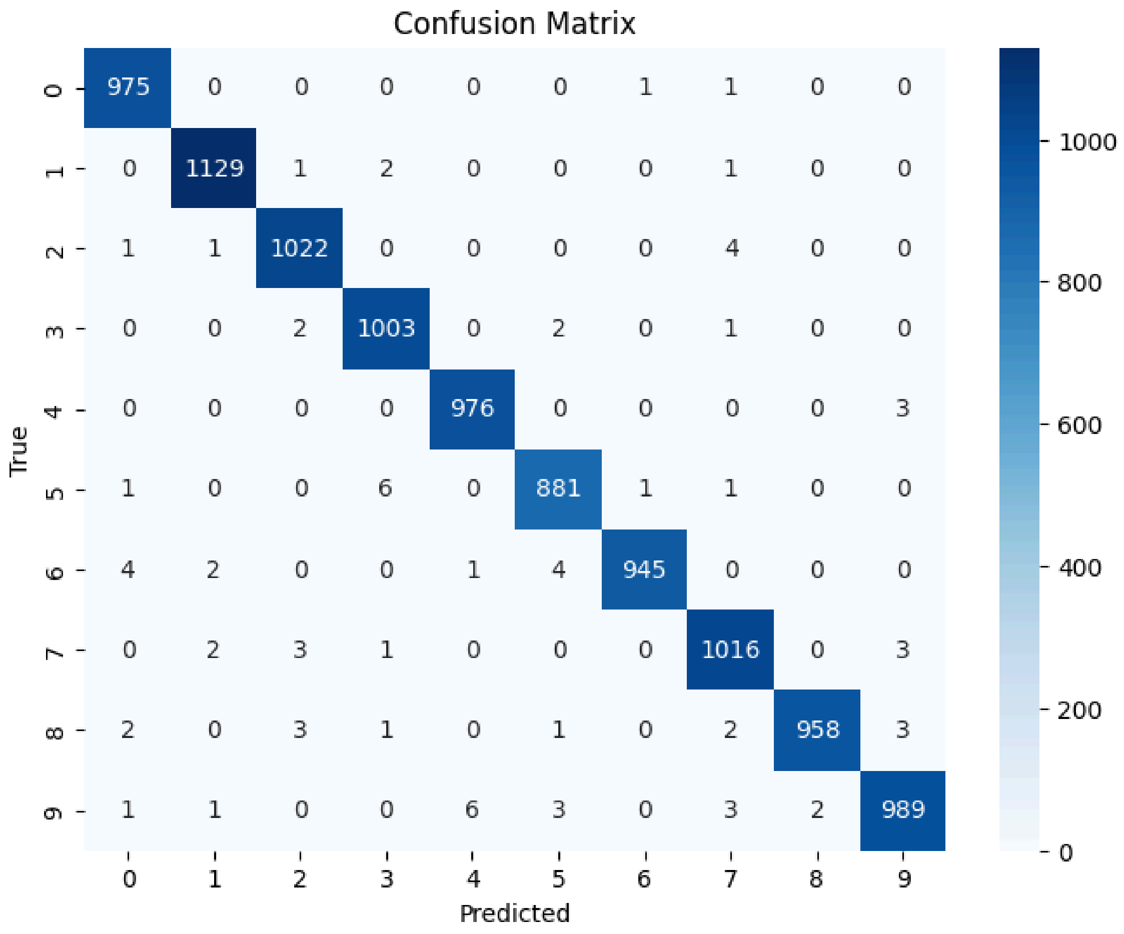
```
In [ ]: for train_idx, val_idx in kfold.split(x_train):
            fold += 1
            print(f"Fold {fold}:")

            # Split data into training and validation sets
            x_fold_train, x_fold_val = x_train[train_idx], x_train[val_idx]
            y_fold_train, y_fold_val = y_train[train_idx], y_train[val_idx]

            class_labels = [str(i) for i in range(10)]

            # Create and compile the model
            model = create_model()
            model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',

            # Train the model
            history = model.fit(x_fold_train, y_fold_train, epochs=5, batch_size=64

            # Evaluate the model
            _, accuracy = model.evaluate(x_test, y_test, verbose=0)
            accuracies.append(accuracy)
            print(f"Test Accuracy for Fold {fold}: {accuracy}")

            # Confusion Matrix
            y_pred = np.argmax(model.predict(x_test), axis=-1)
            conf_matrix = confusion_matrix(y_test, y_pred)
            conf_matrices.append(conf_matrix)
            print("Confusion Matrix:")
            print(conf_matrix)
            plt.figure(figsize=(8, 6))
            sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels
            plt.xlabel("Predicted")
            plt.ylabel("True")
            plt.title("Confusion Matrix")
            plt.show()
```

```
 [   3    0    2    1    0    0    3    0  963    2]
 [   0    2    0    0    5    4    0    2    8  988]]
```



In [ ]:
```
# Average accuracy
print(f"\nAverage Test Accuracy: {np.mean(accuracies)}")

# Average Confusion Matrix
avg_conf_matrix = np.mean(conf_matrices, axis=0)
print("\nAverage Confusion Matrix:")
print(avg_conf_matrix)
# Define class labels for visualization
class_labels = [str(i) for i in range(10)]

# Function to plot confusion matrix
def plot_confusion_matrix(conf_matrix):
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix.astype(int), annot=True, cmap="Blues", xticklab
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix")
    plt.show()

# Plot average confusion matrix
plot_confusion_matrix(avg_conf_matrix)
```

```
Average Test Accuracy: 0.9898599982261658

Average Confusion Matrix:
[[9.7520e+02 4.0000e-01 2.0000e-01 0.0000e+00 0.0000e+00 2.0000e-01
  1.6000e+00 1.6000e+00 4.0000e-01 4.0000e-01]
 [0.0000e+00 1.1296e+03 1.0000e+00 2.0000e+00 2.0000e-01 4.0000e-01
  4.0000e-01 1.0000e+00 2.0000e-01 2.0000e-01]
 [1.0000e+00 1.4000e+00 1.0226e+03 8.0000e-01 8.0000e-01 0.0000e+00
  6.0000e-01 4.8000e+00 0.0000e+00 0.0000e+00]
 [0.0000e+00 0.0000e+00 2.2000e+00 1.0038e+03 0.0000e+00 2.4000e+00
  0.0000e+00 1.0000e+00 6.0000e-01 0.0000e+00]
 [2.0000e-01 0.0000e+00 4.0000e-01 2.0000e-01 9.7620e+02 0.0000e+00
  6.0000e-01 2.0000e-01 4.0000e-01 3.8000e+00]
 [1.2000e+00 0.0000e+00 0.0000e+00 6.2000e+00 0.0000e+00 8.8120e+02
  1.0000e+00 1.0000e+00 6.0000e-01 8.0000e-01]
 [4.0000e+00 2.2000e+00 0.0000e+00 0.0000e+00 1.8000e+00 4.2000e+00
  9.4520e+02 0.0000e+00 6.0000e-01 0.0000e+00]
 [2.0000e-01 2.6000e+00 3.8000e+00 1.0000e+00 0.0000e+00 0.0000e+00
  0.0000e+00 1.0164e+03 8.0000e-01 3.2000e+00]
 [2.6000e+00 2.0000e-01 3.0000e+00 1.6000e+00 4.0000e-01 1.2000e+00
  8.0000e-01 2.0000e+00 9.5880e+02 3.4000e+00]
 [1.2000e+00 1.2000e+00 8.0000e-01 8.0000e-01 6.0000e+00 3.4000e+00
  0.0000e+00 3.4000e+00 2.6000e+00 9.8960e+02]]
```



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	975	0	0	0	0	0	1	1	0	0
1	0	1129	1	2	0	0	0	1	0	0
2	1	1	1022	0	0	0	0	4	0	0
3	0	0	2	1003	0	2	0	1	0	0
4	0	0	0	0	976	0	0	0	0	3
5	1	0	0	6	0	881	1	1	0	0
6	4	2	0	0	1	4	945	0	0	0
7	0	2	3	1	0	0	0	1016	0	3
8	2	0	3	1	0	1	0	2	958	3
9	1	1	0	0	6	3	0	3	2	989