```
pip install adversarial-robustness-toolbox
```

```
    Requirement already satisfied: adversarial-robustness-toolbox in /usr/local/lib/python3.10/dist-packages (
    Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-
    Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-r
    Requirement already satisfied: scikit-learn<1.2.0,>=0.22.2 in /usr/local/lib/python3.10/dist-packages (fro
    Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness
    Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-rob
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustnes
    Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from sciki
```

Pre-trained model

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical

# Load pre-trained model
model = load_model('/content/drive/MyDrive/ColabNotebooks/mnist_model.h5')  # Update with the path to your mode

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
# Reshape the data to fit the model
x_train = x_train.reshape(60000, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(10000, 28, 28, 1).astype('float32') / 255
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test_categorical = to_categorical(y_test, 10)

# Get the model's predictions on the test data
predictions = model.predict(x_test)
predicted_labels = np.argmax(predictions, axis=1)


nb_correct_pred = np.sum(predicted_labels == y_test)

print("Original test data:")
print("Correctly classified: {}".format(nb_correct_pred))
print("Incorrectly classified: {}".format(len(x_test)-nb_correct_pred))

# Evaluate the model on the test data
loss, accuracy = model.evaluate(x_test, y_test_categorical, verbose=0)
print(f'Test accuracy: {accuracy:.3f}')

# Save only the examples that the model identifies correctly
correct_indices = predicted_labels == y_test
correct_examples = x_test[correct_indices]
correct_labels = y_test[correct_indices]

# Save the correct examples and their labels
np.save('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy', correct_examples)
np.save('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy', correct_labels)
```

```
    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training_v1.py:2359: UserWarning: `Model.state_up
      updates=self.state_updates,
    Original test data:
    Correctly classified: 9834
    Incorrectly classified: 166
    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training_v1.py:2335: UserWarning: `Model.state_up
      updates = self.state_updates
    Test accuracy: 0.983
```

```
 # For clean examples
y_pred_clean = np.argmax(model.predict(correct_examples), axis=1)
accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
```

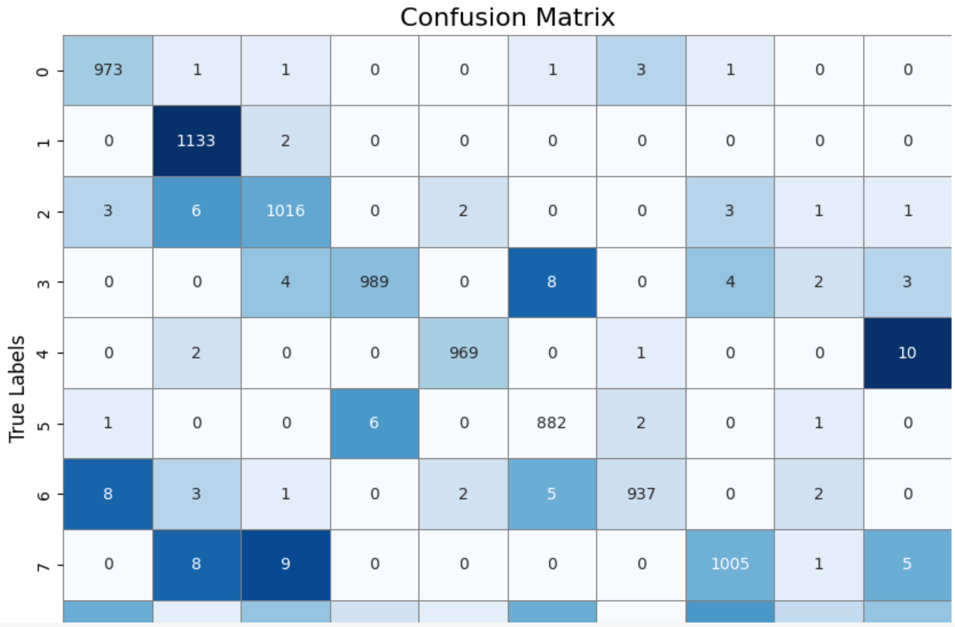
```
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
```

```
    Accuracy on clean data: 100.00%
```

```
# Count the occurrences of each label in the test set
unique, counts = np.unique(correct_labels, return_counts=True)
correct_labels_counts = dict(zip(unique, counts))
print("correct_labels_counts:", correct_labels_counts)
for label, count in correct_labels_counts.items():
    print(f"Label {label}: {count}")
```

```
    correct_labels_counts: {0: 973, 1: 1133, 2: 1016, 3: 989, 4: 969, 5: 882, 6: 937, 7: 1005, 8: 946, 9: 984}
    Label 0: 973
    Label 1: 1133
    Label 2: 1016
    Label 3: 989
    Label 4: 969
    Label 5: 882
    Label 6: 937
    Label 7: 1005
    Label 8: 946
    Label 9: 984
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# Calculate the confusion matrix
cm = confusion_matrix(y_test, predicted_labels,labels=range(10))

# Create a mask for the diagonal elements
mask = np.eye(len(cm), dtype=bool)

# Set up the matplotlib figure
fig, ax = plt.subplots(figsize=(10, 8))

# Plot the heatmap for off-diagonal elements using the mask
# Use a professional color palette like 'Blues'
sns.heatmap(cm, mask=mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey

# Plot the heatmap for diagonal elements using the inverse of the mask
# Use the same color palette for consistency
sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='gre

# Labels, title and ticks
label_names = [f'{i}' for i in range(10)]
ax.set_xlabel('Predicted Labels', fontsize=12)
ax.set_ylabel('True Labels', fontsize=12)
ax.set_title('Confusion Matrix', fontsize=16)
ax.set_xticklabels(label_names)
ax.set_yticklabels(label_names)

plt.savefig('model.png', bbox_inches='tight')
plt.show()
```



```
predictions.shape
```

```
(10000, 10)
```

```
# Count the occurrences of each label in the correct_labels
unique, counts = np.unique(predicted_labels, return_counts=True)
predicted_labels_counts = dict(zip(unique, counts))
print("correct_predicted set label counts:", predicted_labels_counts)
for label, count in predicted_labels_counts.items():
    print(f"Label {label}: {count}")

total_predicted_labels_count = len(predicted_labels)

# Display the total count of all labels
print(f"Total count of all total_predicted_labels_count combined in the MNIST dataset: {total_predicted_labels_
```

```
correct_predicted set label counts: {0: 992, 1: 1157, 2: 1037, 3: 999, 4: 983, 5: 904, 6: 943, 7: 1024, 8:
Label 0: 992
Label 1: 1157
Label 2: 1037
Label 3: 999
Label 4: 983
Label 5: 904
Label 6: 943
Label 7: 1024
Label 8: 954
Label 9: 1007
Total count of all total_predicted_labels_count combined in the MNIST dataset: 10000
```

FastGradientMethod

Load the saved examples Choose attack FGM Apply without target,with target, muktiple attacks without target and with target

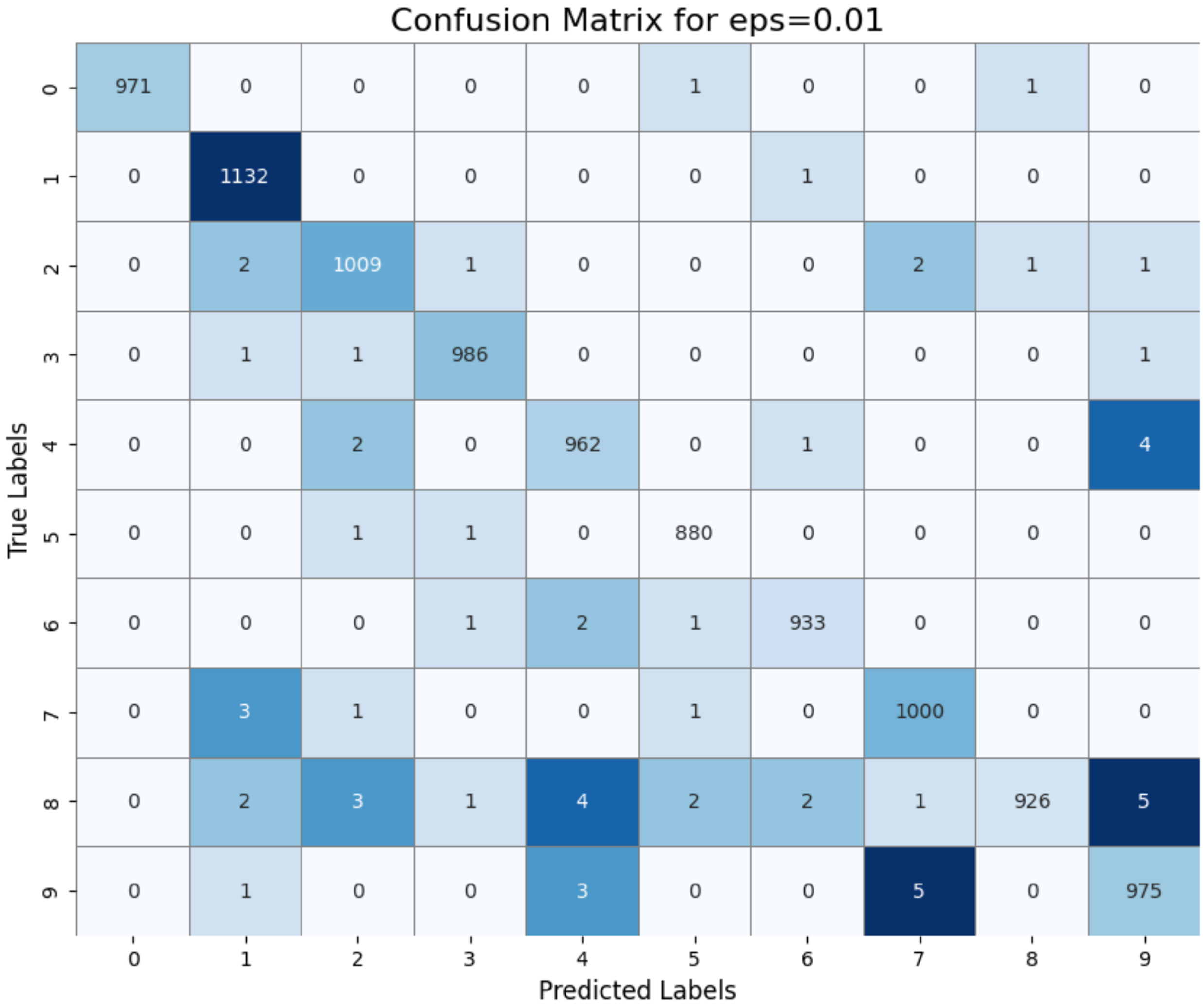
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import KerasClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Disable eager execution (necessary for ART with TensorFlow v1)
tf.compat.v1.disable_eager_execution()

# Load your trained model
```

```
model = load_model('/content/drive/MyDrive/ColabNotebooks/mnist_model.h5')

# Load the saved correct examples and their labels
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy')

# Preprocess the examples
#correct_examples = correct_examples.astype('float32') / 255

# Wrap the model with ART KerasClassifier
classifier = KerasClassifier(model=model, clip_values=(0, 1))

 # For clean examples
y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training_v1.py:2359: UserWarning: `Model.state_up
  updates=self.state_updates,
Accuracy on clean data: 100.00%
```

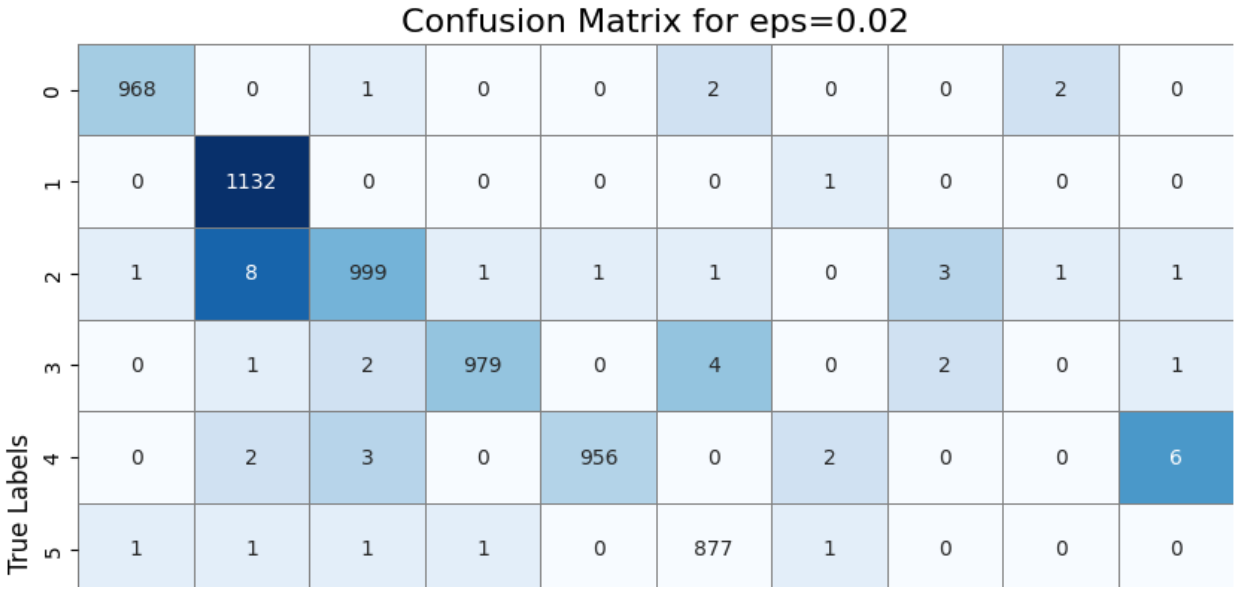
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import KerasClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Disable eager execution (necessary for ART with TensorFlow v1)
tf.compat.v1.disable_eager_execution()

# Load your trained model
model = load_model('/content/drive/MyDrive/ColabNotebooks/mnist_model.h5')

# Load the saved correct examples and their labels
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy')

# Preprocess the examples
# correct_examples = correct_examples.astype('float32') / 255

# Wrap the model with ART KerasClassifier
classifier = KerasClassifier(model=model, clip_values=(0, 1))

 # For clean examples
y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Define the range of eps values
eps_range = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

# Initialize a DataFrame to store results
results_df = pd.DataFrame(columns=['eps', 'total_correct', 'total_adv', 'correct_adv_counts'])

# Loop over the eps values
for eps in eps_range:
    # Define the attack with the current eps
    attack = FastGradientMethod(classifier, eps=eps)

    # Apply the attack to generate adversarial examples
    x_adv = attack.generate(x=correct_examples)

    # Predict the labels of the adversarial examples
    y_adv = np.argmax(classifier.predict(x_adv), axis=1)


    nb_correct_adv_pred = np.sum(y_adv == correct_labels)

    print(f"Adversarial test data: eps:{eps}")
    print("Correctly classified: {}".format(nb_correct_adv_pred))
    print("Incorrectly classified: {}".format(len(correct_examples)-nb_correct_adv_pred))

    # For clean examples
    y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
    accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
    print(f"accuracy_clean:{accuracy_clean}")
    # Inside the loop for each eps
    # You have already calculated this for adversarial examples:
    # nb_correct_adv_pred = np.sum(y_adv == correct_labels)
    # Now calculate the accuracy for adversarial examples
    accuracy_adv = nb_correct_adv_pred / len(correct_labels)

    print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
    print(f"Adversarial test data: eps:{eps}")
    print(f"Accuracy on adversarial examples: {accuracy_adv * 100:.2f}%")



    # Count the occurrences of each label in the adversarial predictions
    unique_adv, counts_adv = np.unique(y_adv, return_counts=True)
    adv_counts = dict(zip(unique_adv, counts_adv))

    # Calculate the confusion matrix
    cm = confusion_matrix(correct_labels, y_adv, labels=range(10))

    # Draw and save the confusion matrix
    fig, ax = plt.subplots(figsize=(10, 8))

    # Create a mask for the diagonal elements
    mask = np.eye(len(cm), dtype=bool)

    # Plot the heatmap for off-diagonal elements using the mask
    # Use a professional color palette like 'Blues'
    sns.heatmap(cm, mask=mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='

    # Plot the heatmap for diagonal elements using the inverse of the mask
    # Use the same color palette for consistency
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor=
    # Labels, title and ticks
    label_names = [f'{i}' for i in range(10)]
    ax.set_xlabel('Predicted Labels', fontsize=12)
    ax.set_ylabel('True Labels', fontsize=12)
    ax.set_title(f'Confusion Matrix for eps={eps}', fontsize=16)
    ax.set_xticklabels(label_names)
    ax.set_yticklabels(label_names)


    image_filename = f'confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()


    # Save the results in the DataFrame
    results_df = results_df.append({
        'eps': eps,
        'total_correct': len(correct_labels),
        'total_adv': len(y_adv),
        'correct_adv_counts': adv_counts
    }, ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('/content/drive/MyDrive/ColabNotebooks/withouttruelabel_adv_results.csv', index=False)

# Print the DataFrame
print(results_df)
```

```
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```

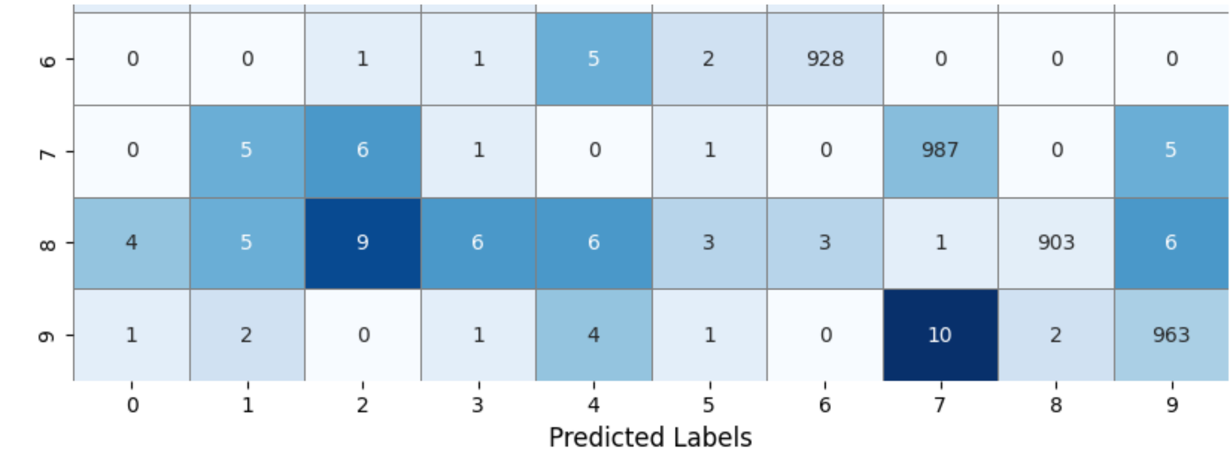


	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



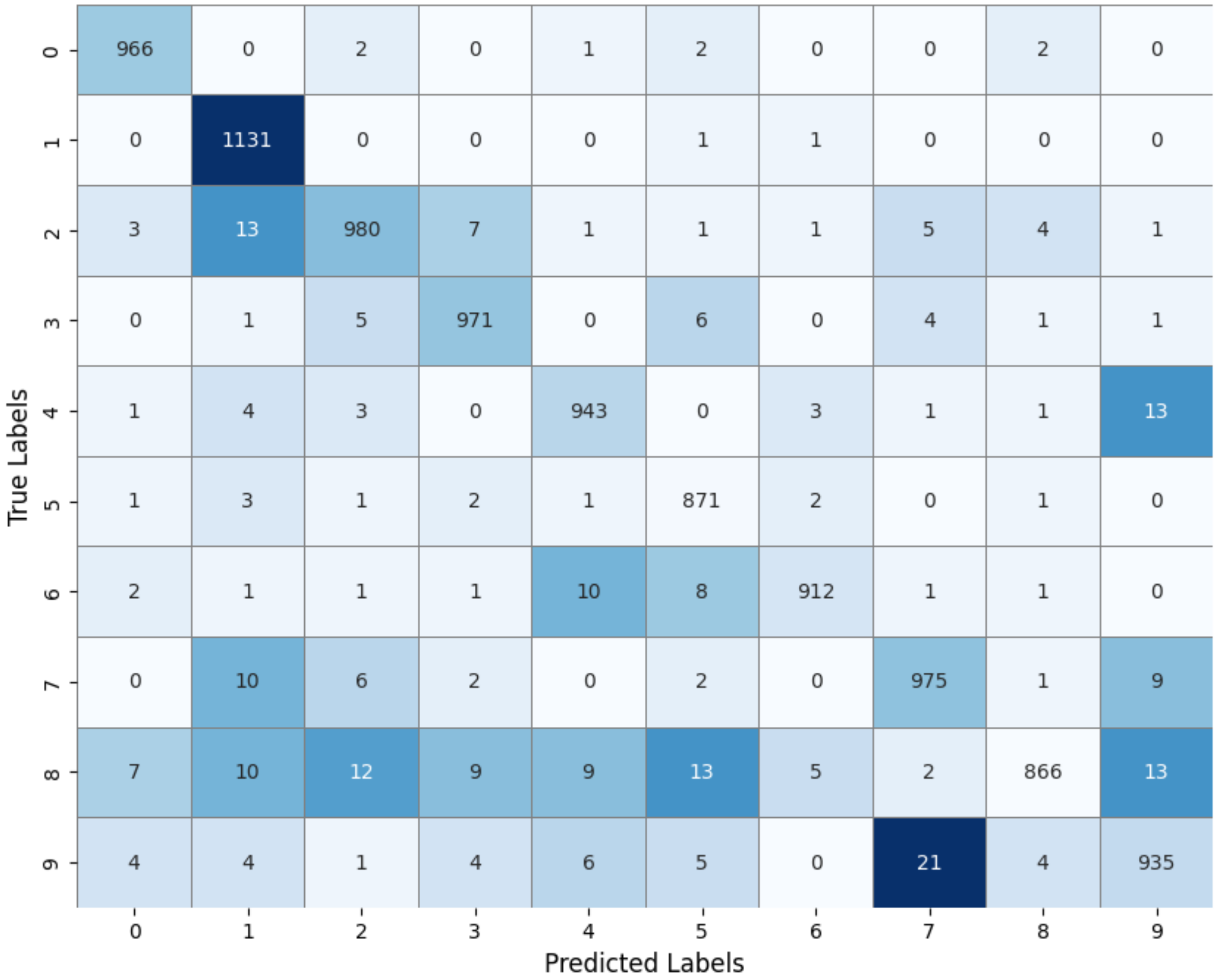
	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0



	0	1	2	3	4	5	6	7	8	9
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remov
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```

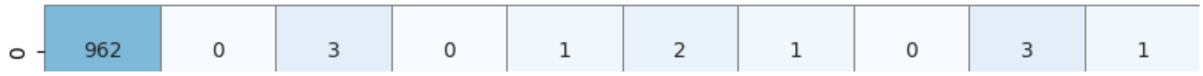
Confusion Matrix for eps=0.03



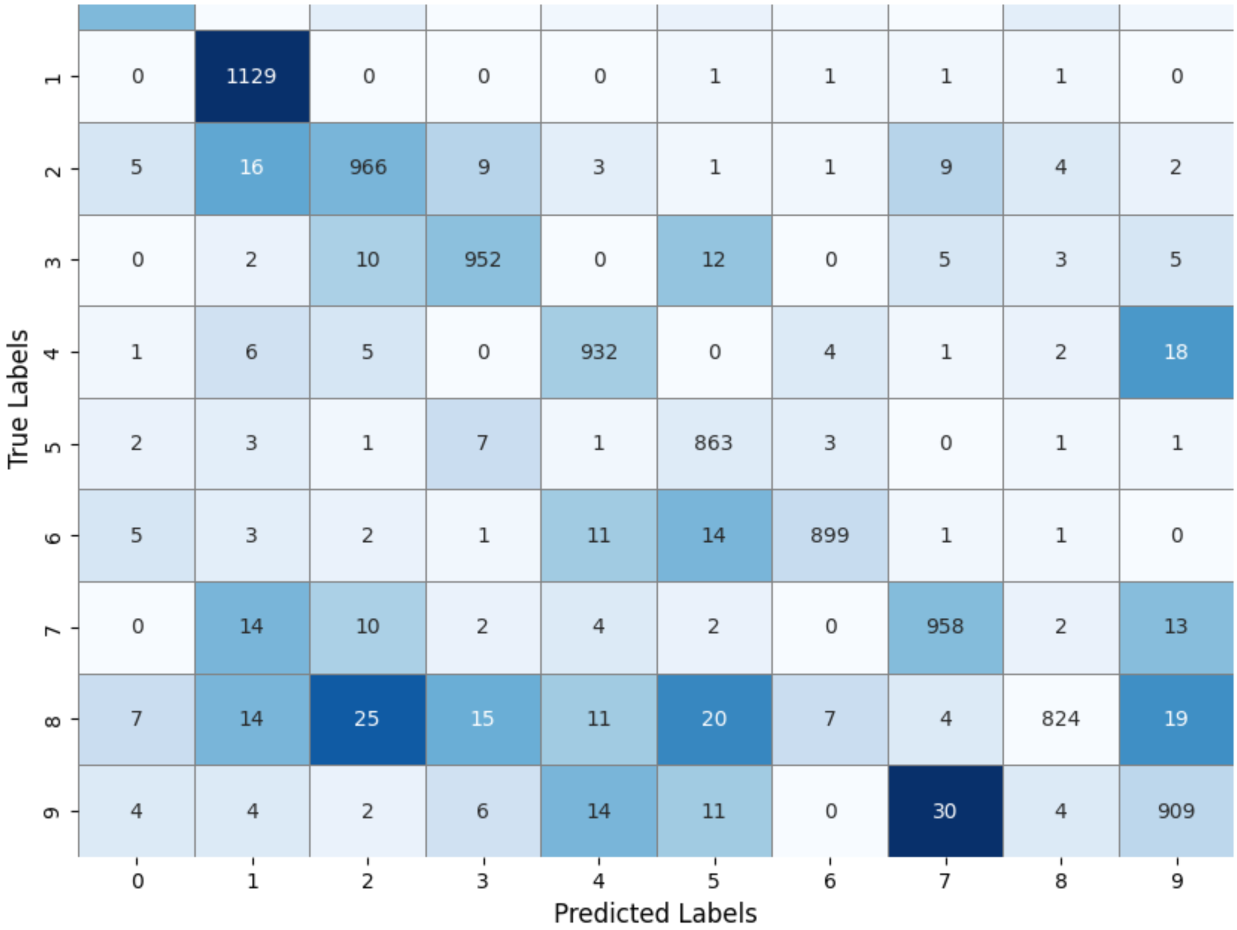
	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remov
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

Confusion Matrix for eps=0.04



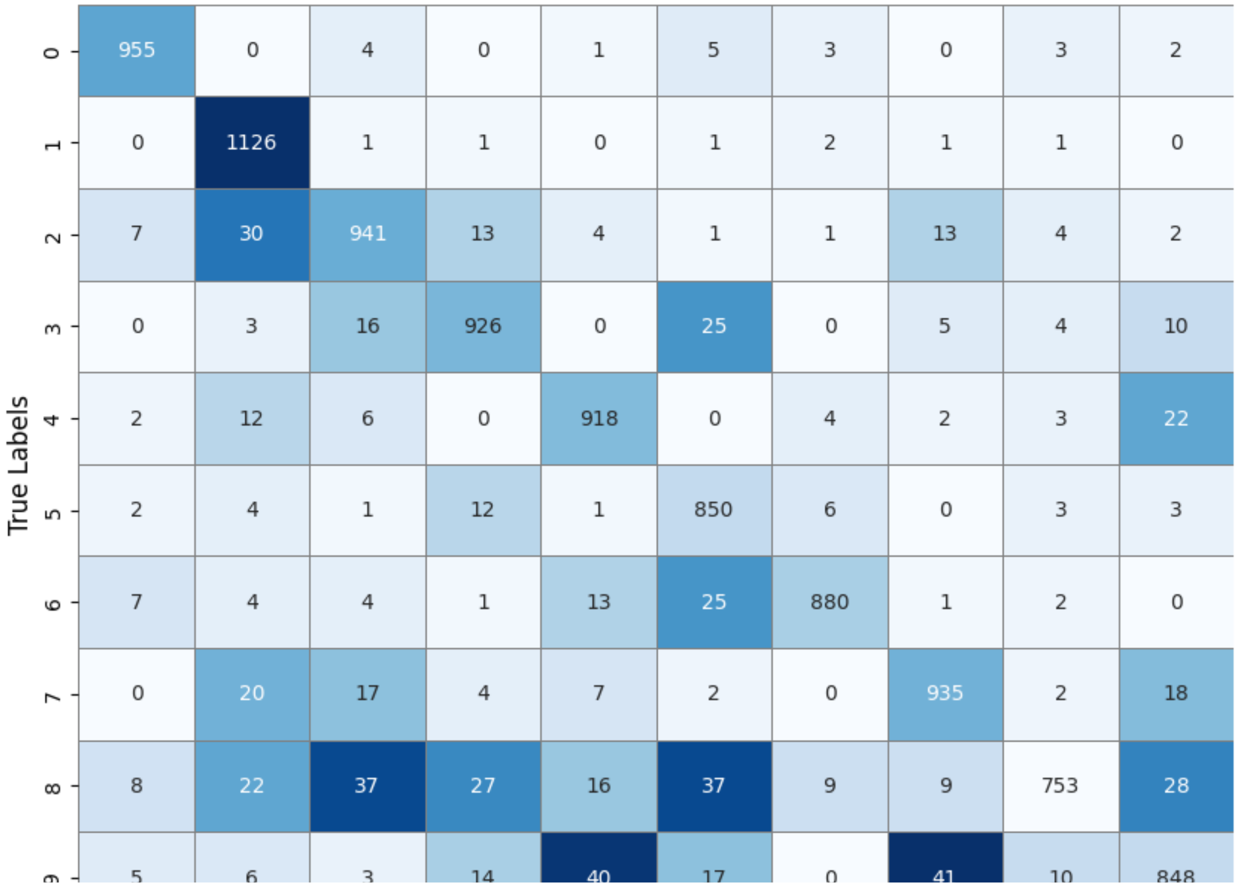
	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1



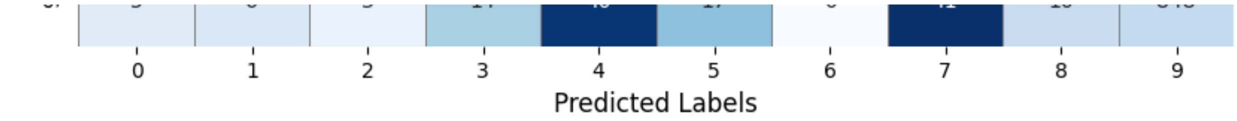
	0	1	2	3	4	5	6	7	8	9
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```

Confusion Matrix for eps=0.05

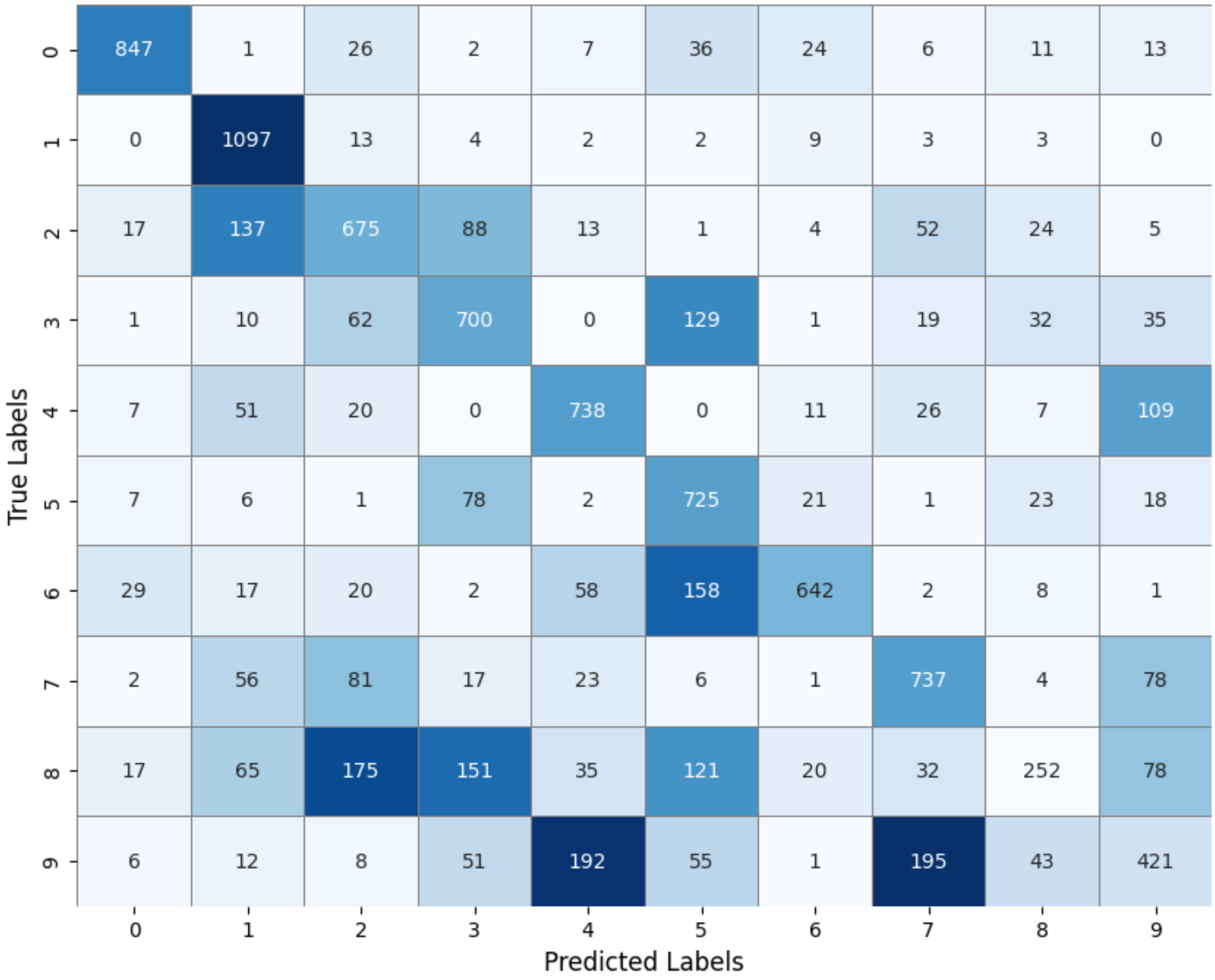


	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848



```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remov
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

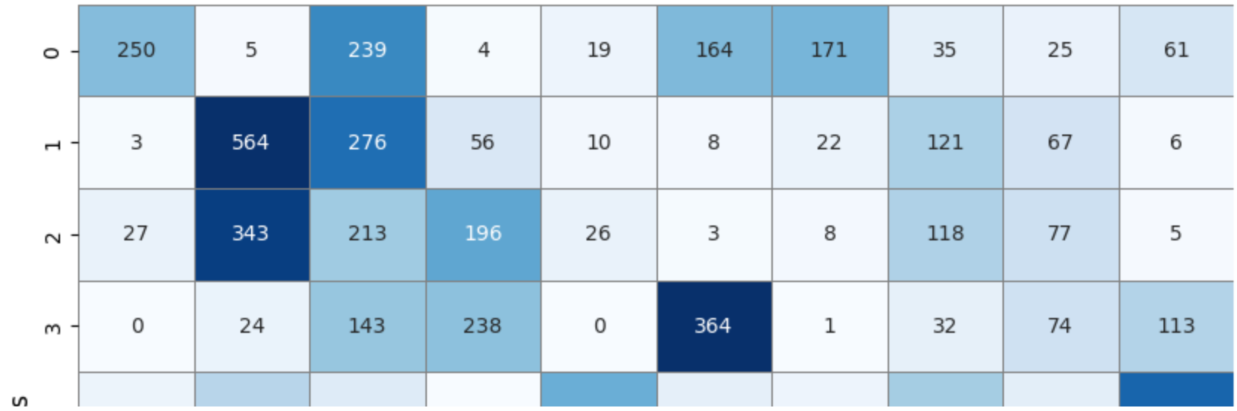
Confusion Matrix for eps=0.1



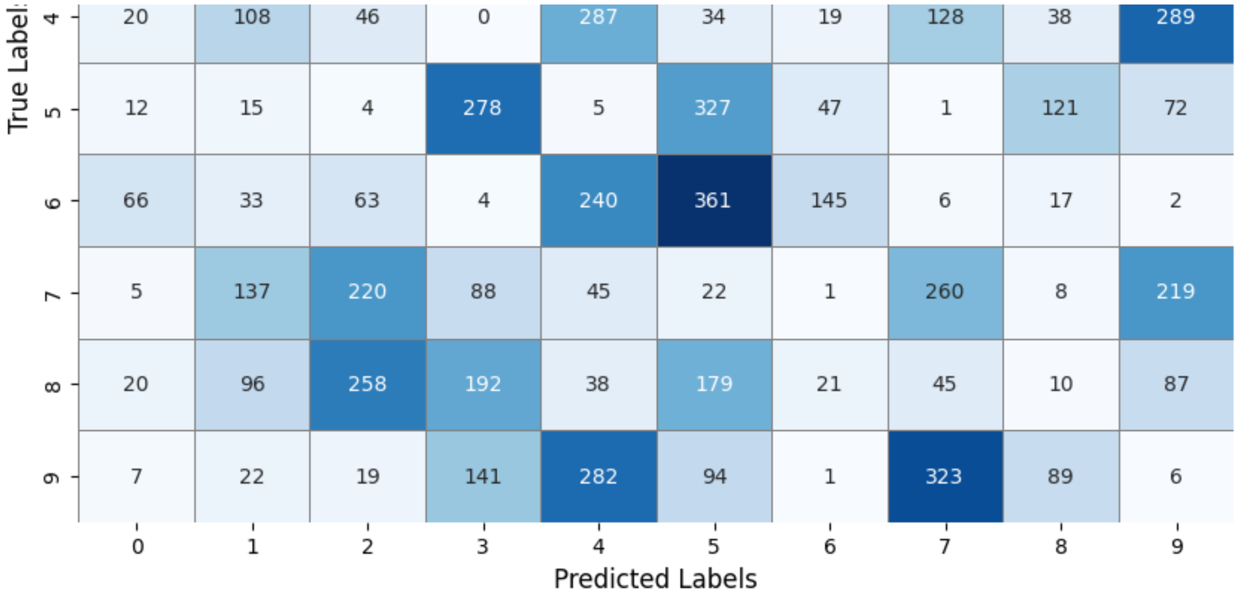
	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remov
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

Confusion Matrix for eps=0.2

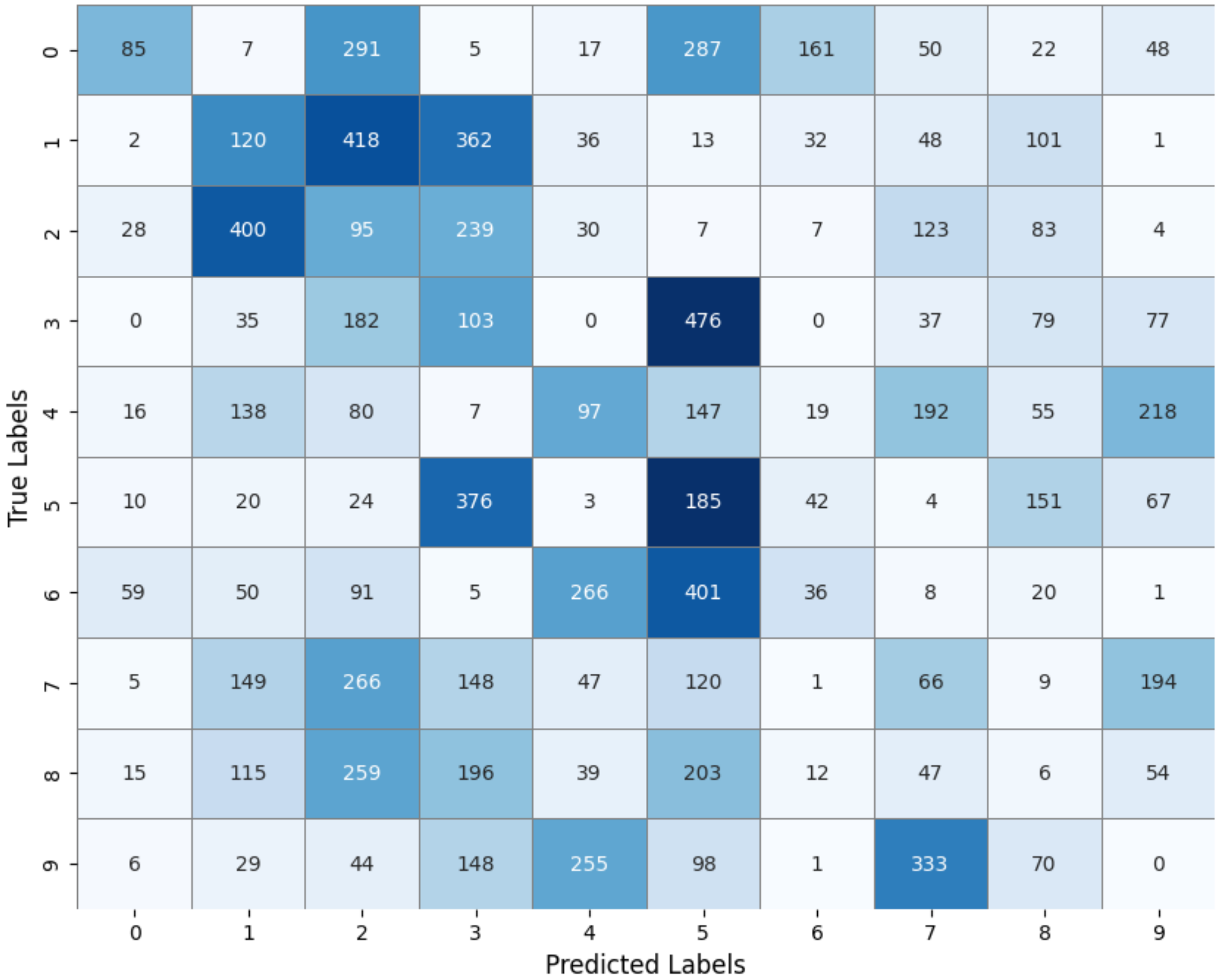


0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113



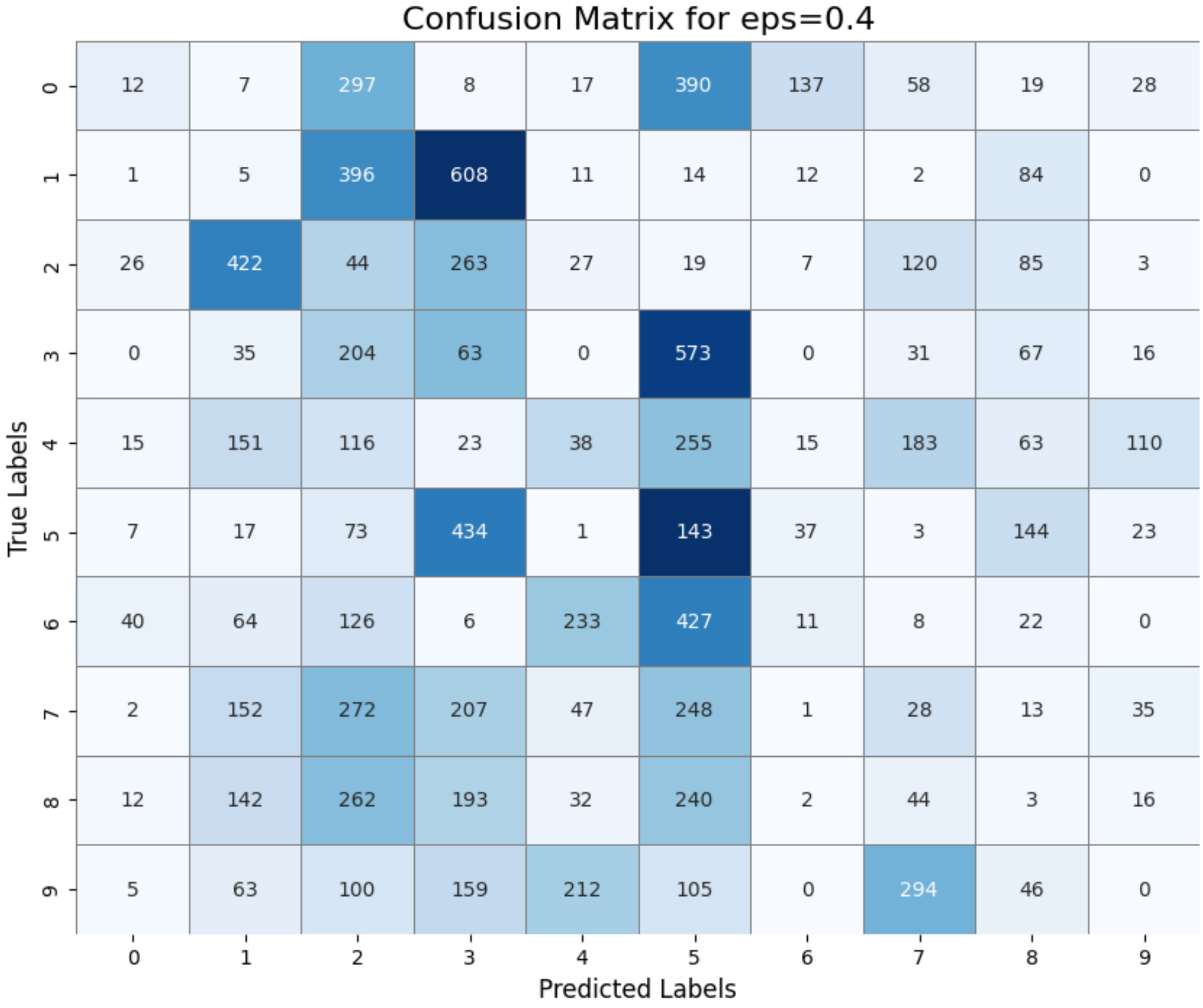
```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

Confusion Matrix for eps=0.3



```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
```

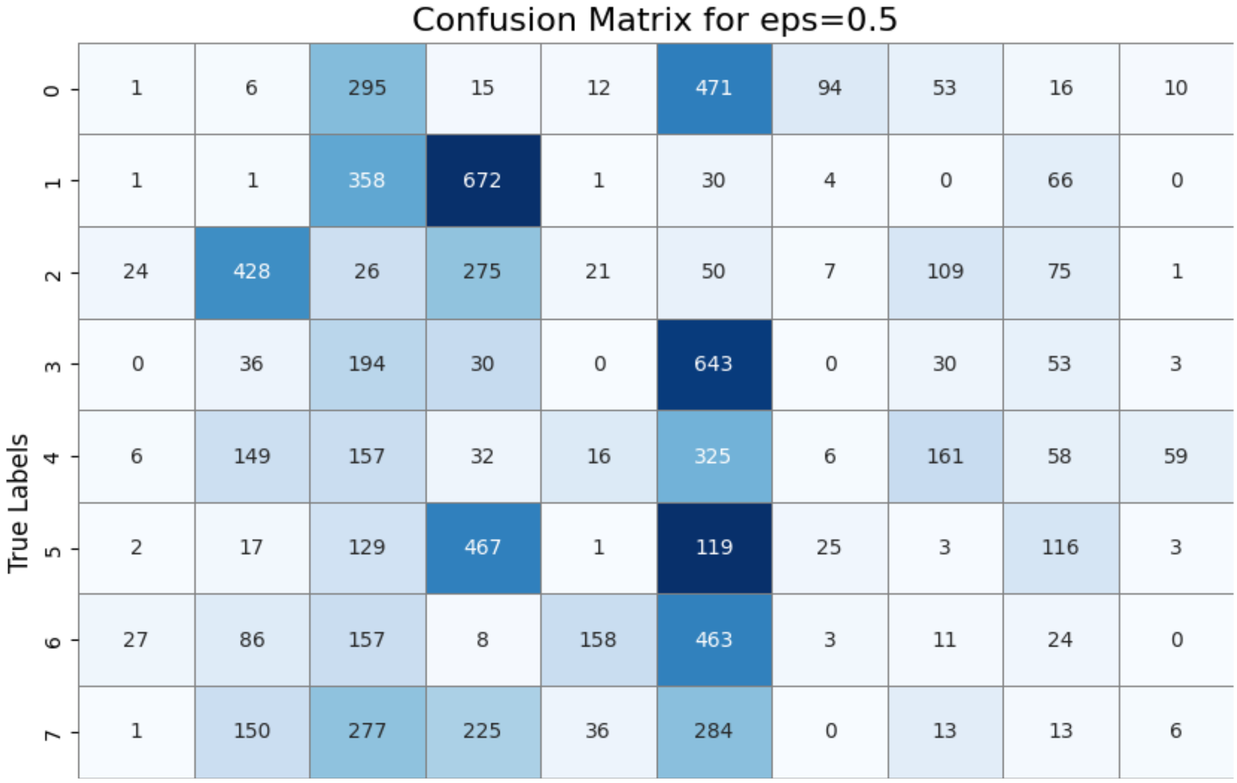
```
Accuracy on adversarial examples: 3.53%
```



Confusion Matrix for eps=0.4

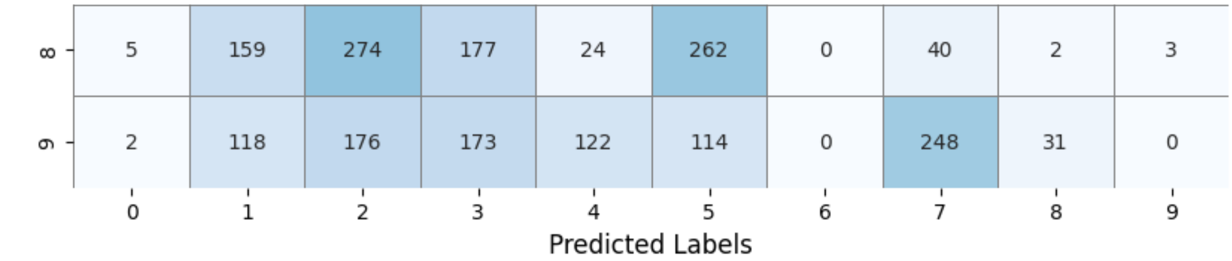
	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```



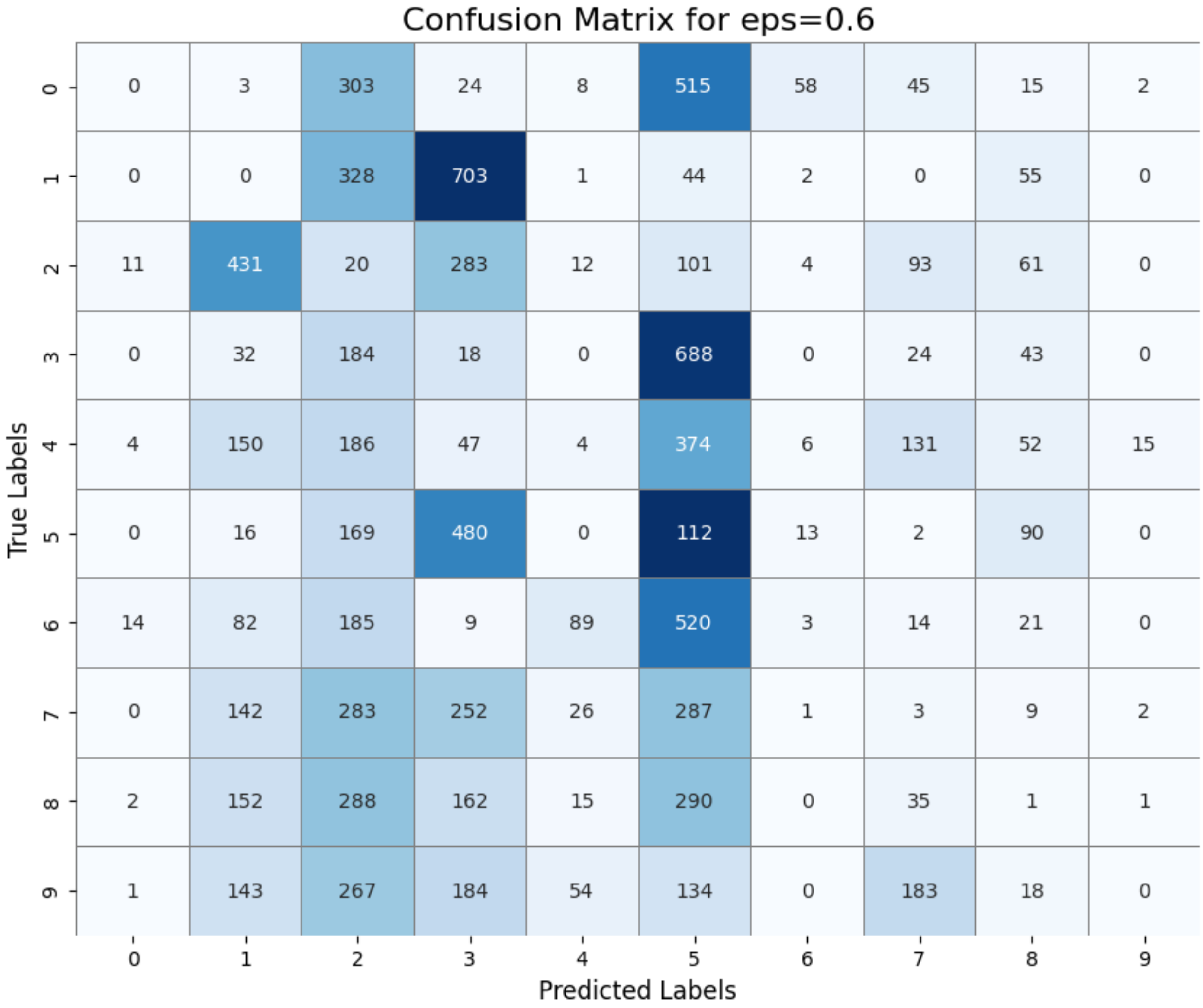
Confusion Matrix for eps=0.5

0	1	6	295	15	12	471	94	53	16	10
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6



	0	1	2	3	4	5	6	7	8	9
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remo
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```



Confusion Matrix for eps=0.6

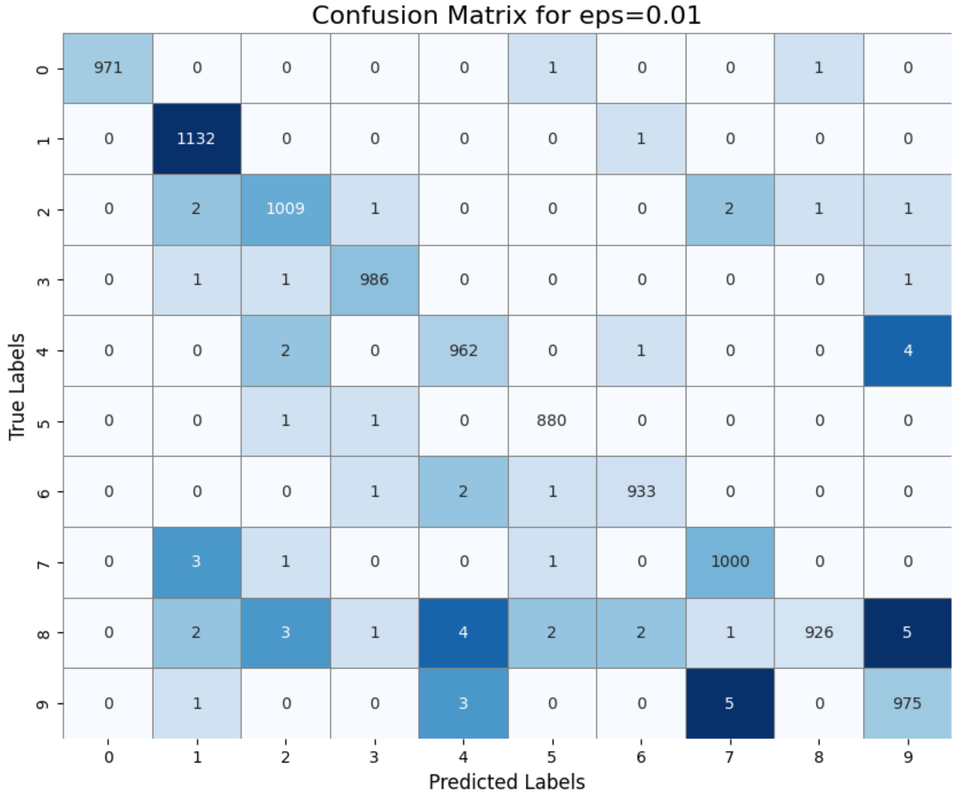
	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
     eps  total_correct  total_adv  \
0   0.01           9834       9834   
1   0.02           9834       9834   
2   0.03           9834       9834   
3   0.04           9834       9834   
4   0.05           9834       9834   
5   0.10           9834       9834   
6   0.20           9834       9834   
7   0.30           9834       9834   
8   0.40           9834       9834   
9   0.50           9834       9834   
10  0.60           9834       9834   

                                   correct_adv_counts  
0   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...  
1   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...  
2   {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...  
3   {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...  
4   {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...  
5   {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...  
6   {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...  
```

```
7   {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
8   {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
9   {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
10  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
<ipython-input-64-18fe375a8b4b>:107: FutureWarning: The frame.append method is deprecated and will be remov
```

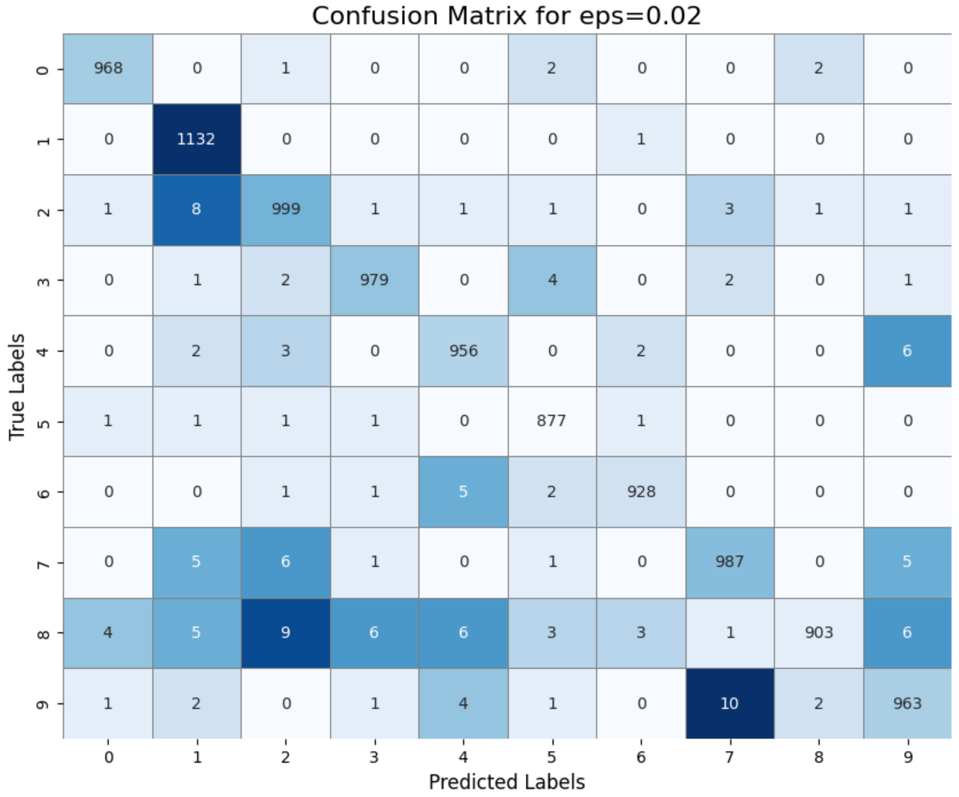
```
# Define the range of eps values
eps_range = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

# Initialize a DataFrame to store results
results_df = pd.DataFrame(columns=['eps', 'total_correct', 'total_adv', 'correct_adv_counts'])

# Loop over the eps values
for eps in eps_range:
    # Define the attack with the current eps
    attack = FastGradientMethod(classifier, eps=eps)

    # Apply the attack to generate adversarial examples
    x_adv = attack.generate(x=correct_examples ,y=correct_labels)

    # Predict the labels of the adversarial examples
    y_adv = np.argmax(classifier.predict(x_adv), axis=1)

    nb_correct_adv_pred = np.sum(y_adv == correct_labels)

    print(f"Adversarial test data: eps:{eps}")
    print("Correctly classified: {}".format(nb_correct_adv_pred))
    print("Incorrectly classified: {}".format(len(correct_examples)-nb_correct_adv_pred))

    # For clean examples
    y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
    accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
    print(f"accuracy_clean:{accuracy_clean}")
    # Inside the loop for each eps
    # You have already calculated this for adversarial examples:
    # nb_correct_adv_pred = np.sum(y_adv == correct_labels)
    # Now calculate the accuracy for adversarial examples
    accuracy_adv = nb_correct_adv_pred / len(correct_labels)

    print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
    print(f"Adversarial test data: eps:{eps}")
    print(f"Accuracy on adversarial examples: {accuracy_adv * 100:.2f}%")



    # Count the occurrences of each label in the adversarial predictions
    unique_adv, counts_adv = np.unique(y_adv, return_counts=True)
    adv_counts = dict(zip(unique_adv, counts_adv))

    # Calculate the confusion matrix
    cm = confusion_matrix(correct_labels, y_adv, labels=range(10))

   # Draw and save the confusion matrix
    fig, ax = plt.subplots(figsize=(10, 8))

    # Create a mask for the diagonal elements
    mask = np.eye(len(cm), dtype=bool)

    # Plot the heatmap for off-diagonal elements using the mask
    # Use a professional color palette like 'Blues'
    sns.heatmap(cm, mask=mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='

    # Plot the heatmap for diagonal elements using the inverse of the mask
    # Use the same color palette for consistency
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor=
    # Labels, title and ticks
    label_names = [f'{i}' for i in range(10)]
    ax.set_xlabel('Predicted Labels', fontsize=12)
    ax.set_ylabel('True Labels', fontsize=12)
    ax.set_title(f'Confusion Matrix for eps={eps}', fontsize=16)
    ax.set_xticklabels(label_names)
    ax.set_yticklabels(label_names)
    image_filename = f'_correct_labels_confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()
    # Save the results in the DataFrame
    results_df = results_df.append({
        'eps': eps,
        'total_correct': len(correct_labels),
        'total_adv': len(y_adv),
        'correct_adv_counts': adv_counts
    }, ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('/content/drive/MyDrive/ColabNotebooks/with_truelabel_adv_results_labels.csv', index=False)

# Print the DataFrame
print(results_df)
```

```
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```

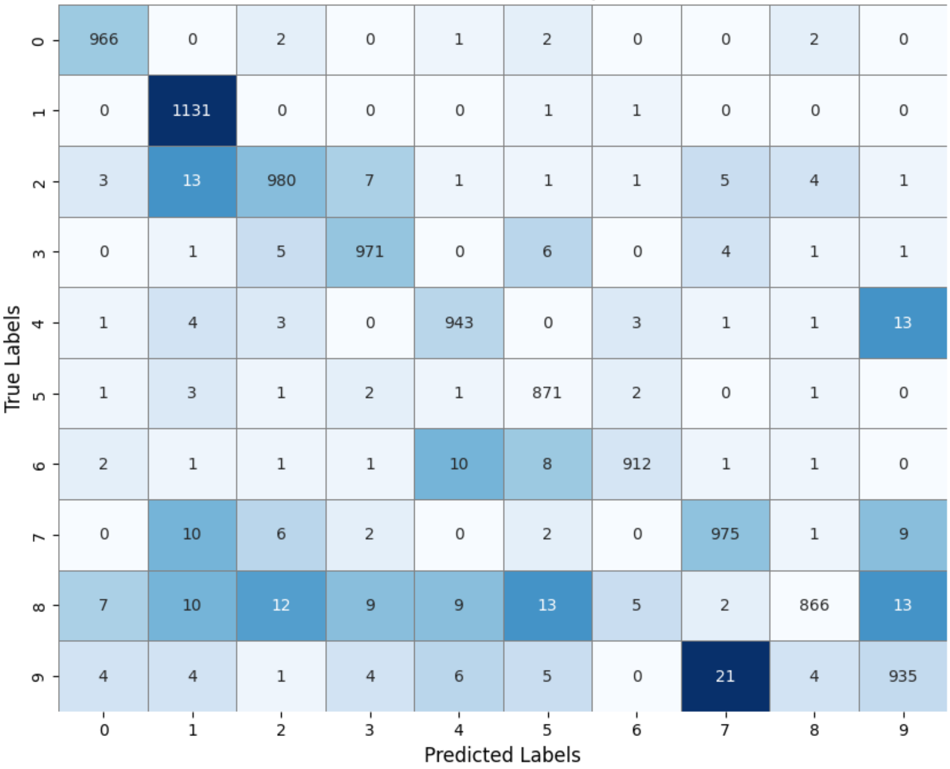


	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
```

```
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```

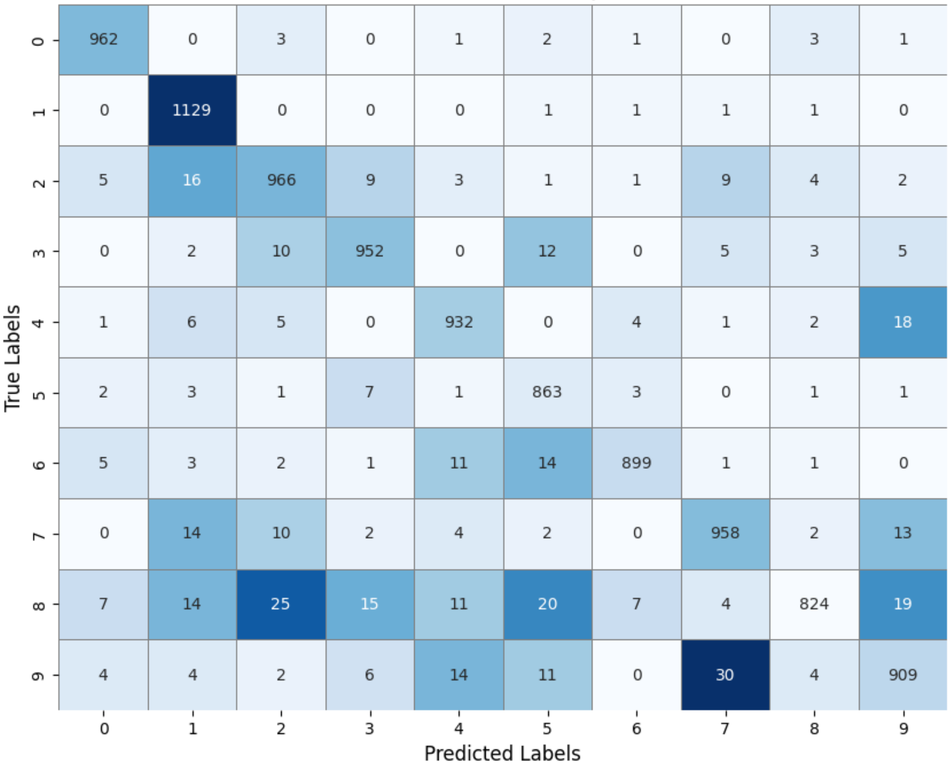
Confusion Matrix for eps=0.03



	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

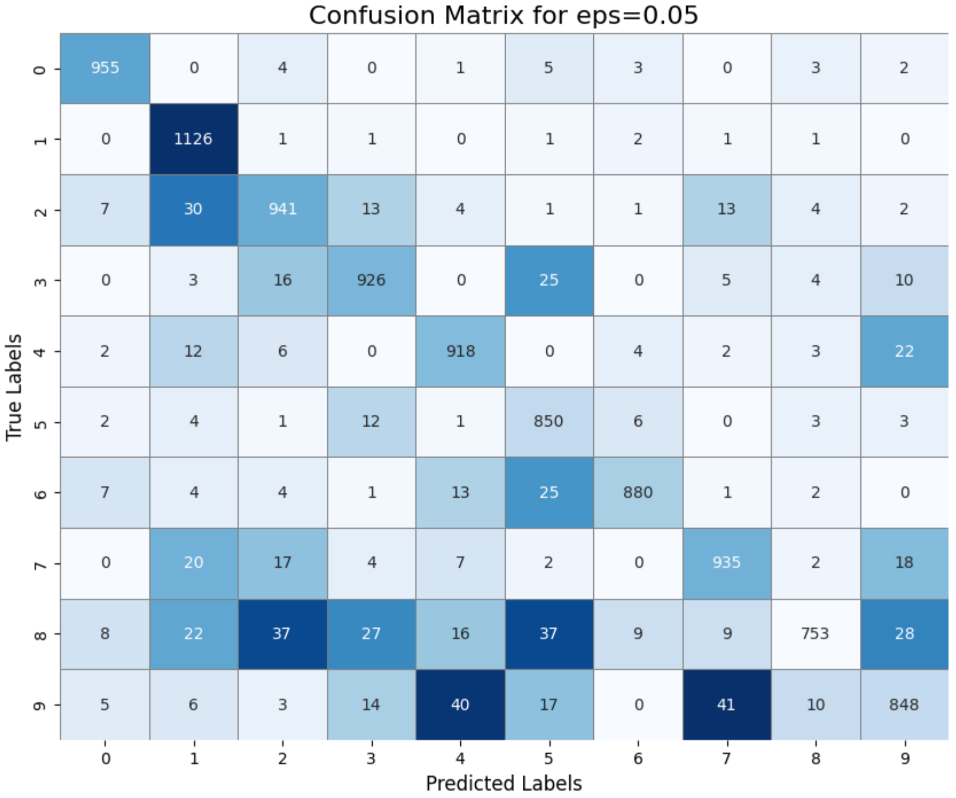
```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

Confusion Matrix for eps=0.04



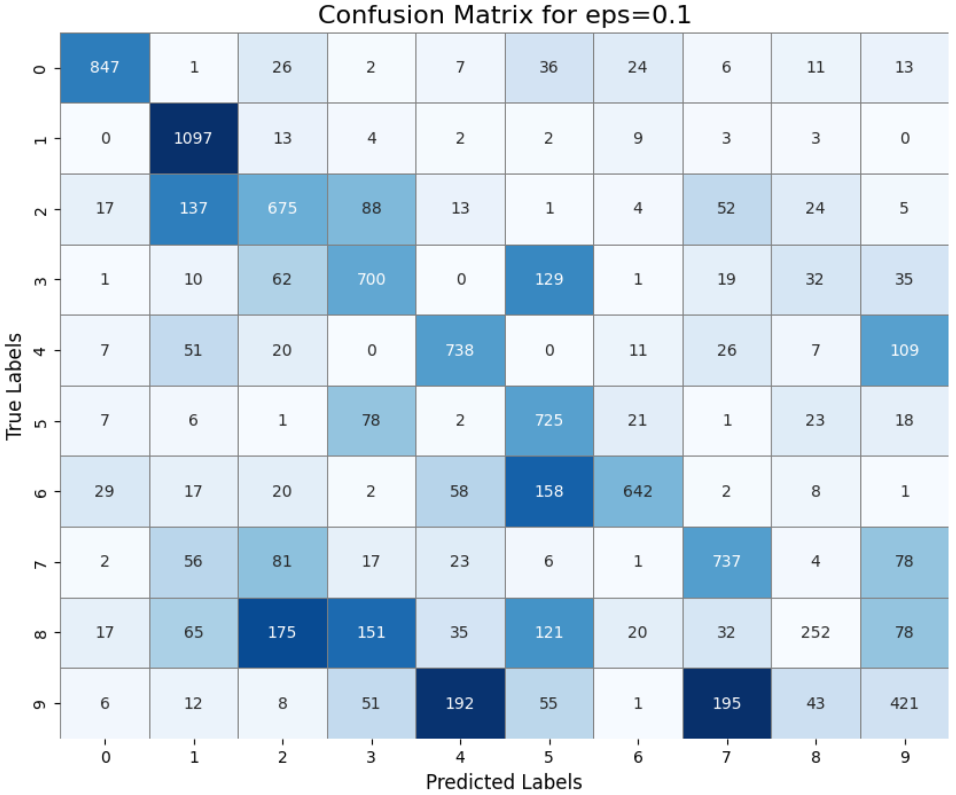
	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```


Confusion Matrix for eps=0.05

	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

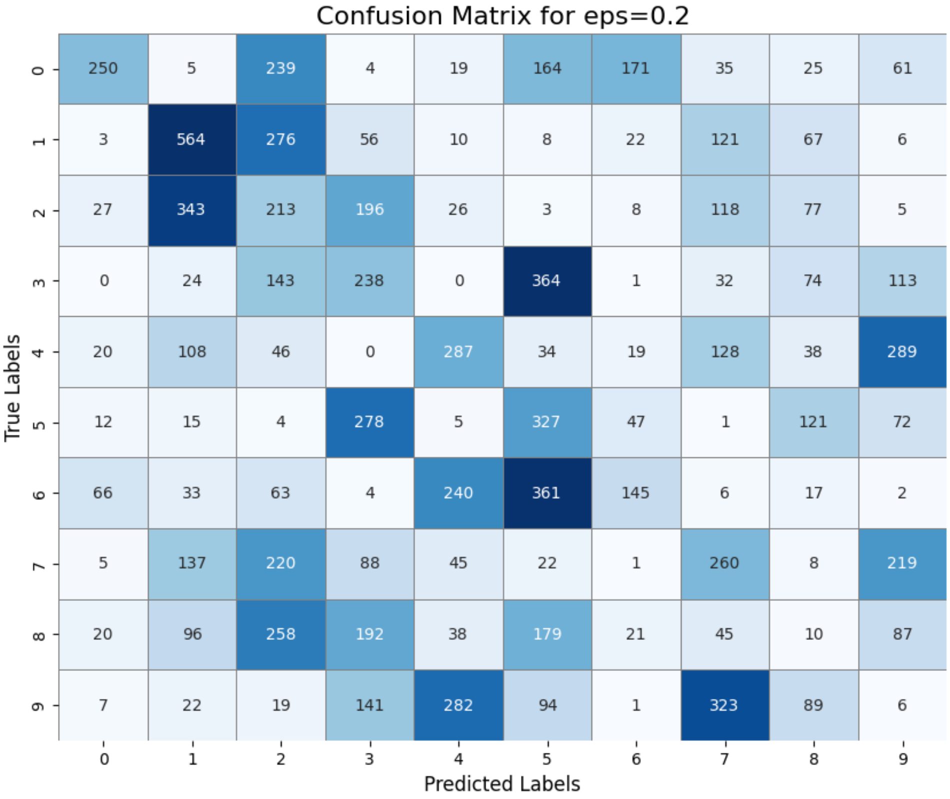
```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```


Confusion Matrix for eps=0.1

	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
```

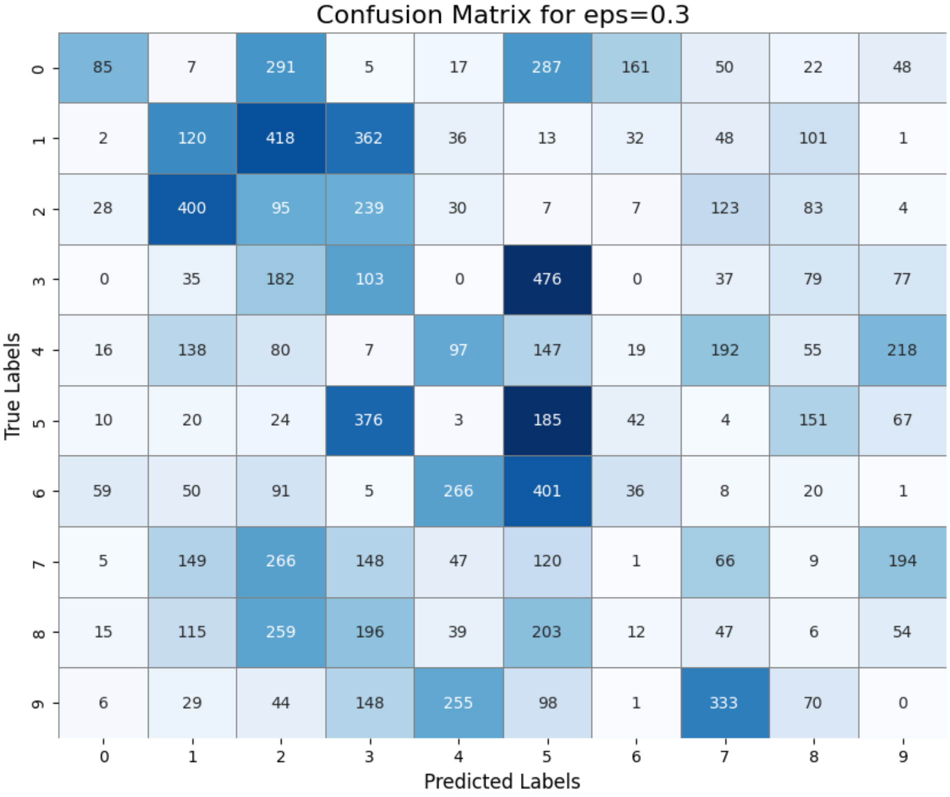
```
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```



Confusion Matrix for eps=0.2

	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

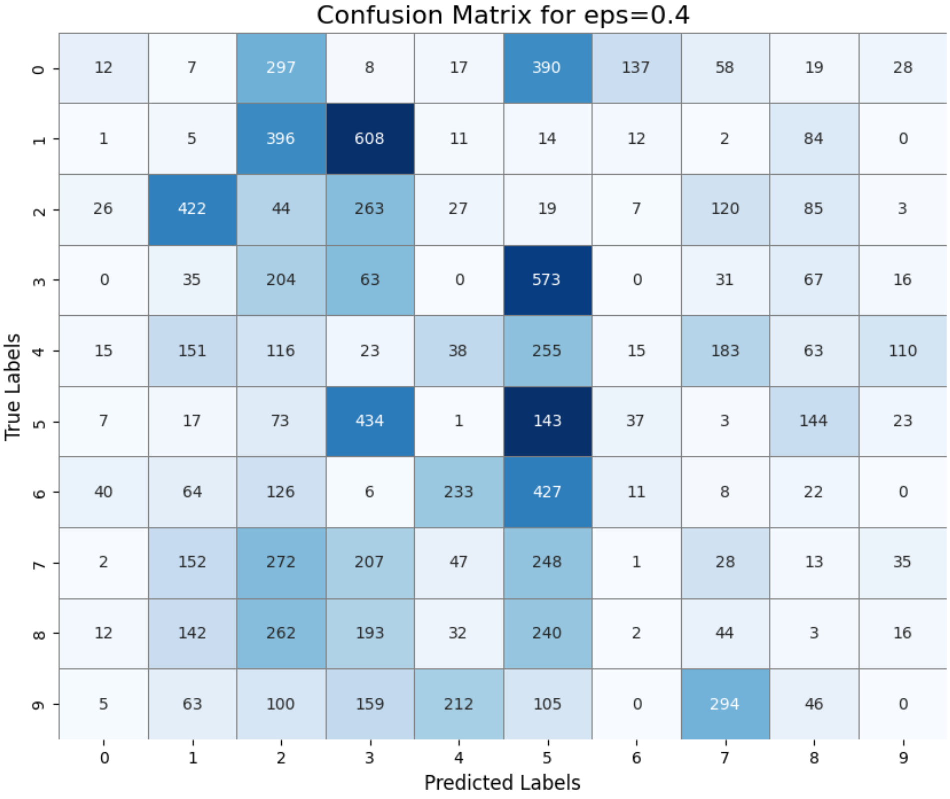
```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```



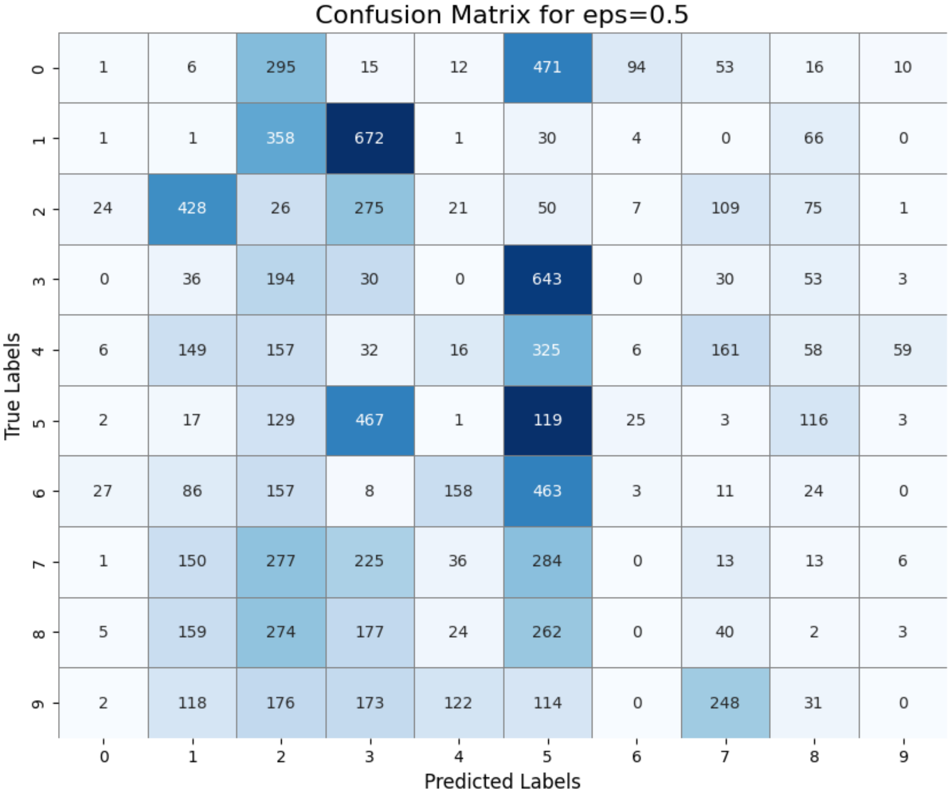
Confusion Matrix for eps=0.3

	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

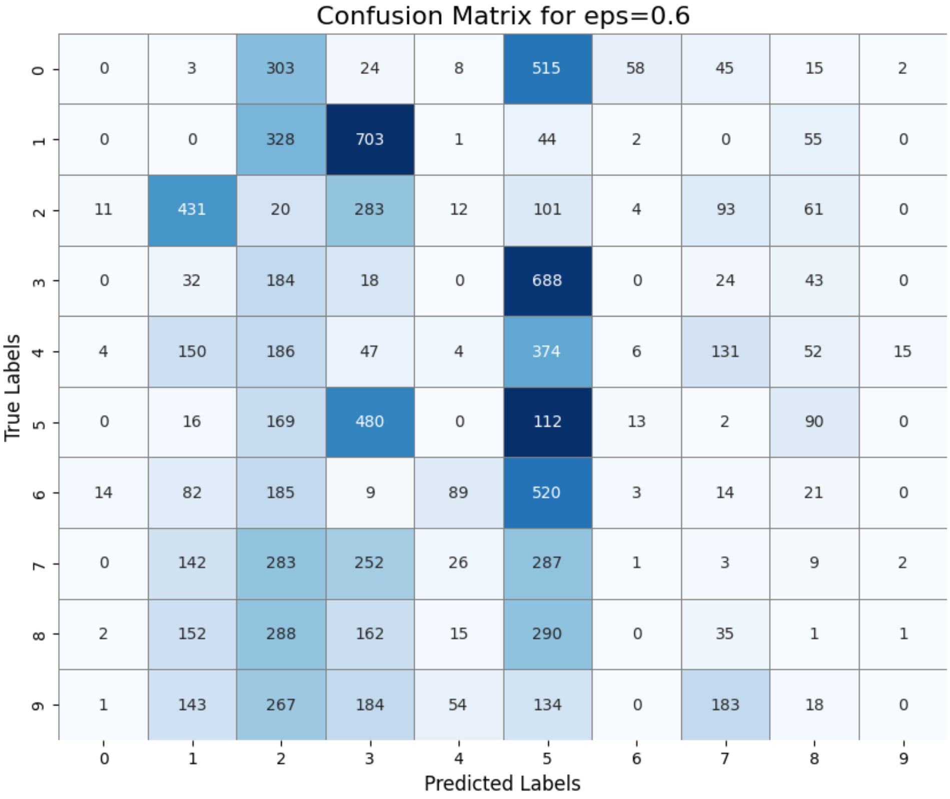

Confusion Matrix for eps=0.4

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```


Confusion Matrix for eps=0.5

```
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
```

```
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

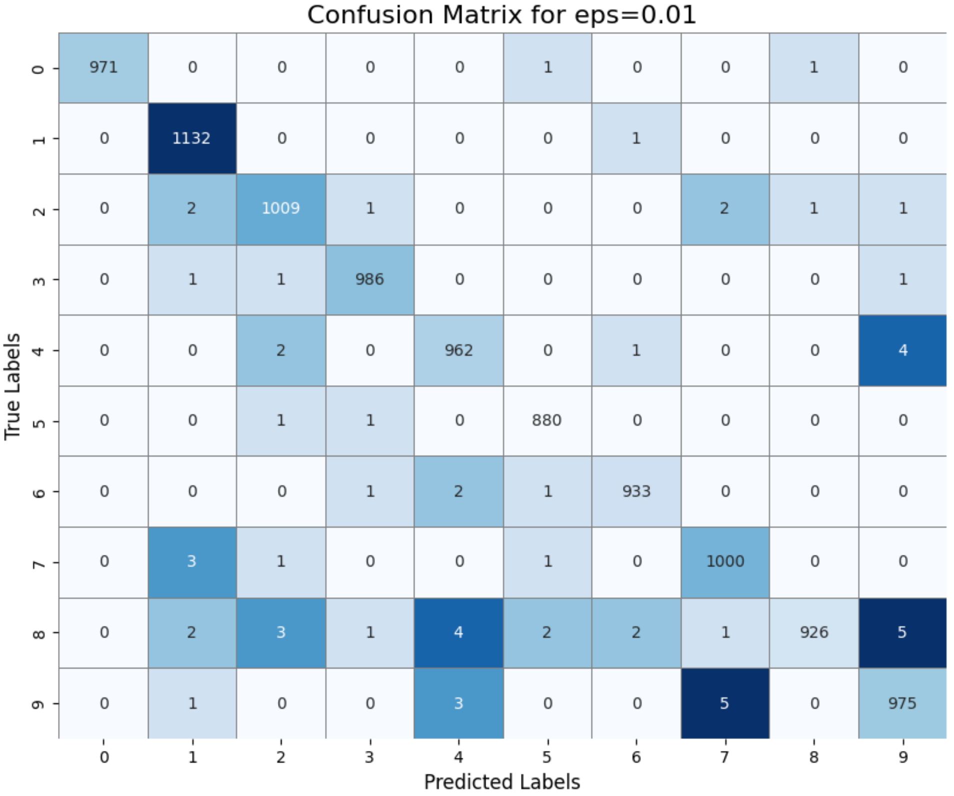

Confusion Matrix for eps=0.6

	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
     eps  total_correct  total_adv  \
0   0.01           9834       9834
1   0.02           9834       9834
2   0.03           9834       9834
3   0.04           9834       9834
4   0.05           9834       9834
5   0.10           9834       9834
6   0.20           9834       9834
7   0.30           9834       9834
8   0.40           9834       9834
9   0.50           9834       9834
10  0.60           9834       9834

                                 correct_adv_counts
0   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
1   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
2   {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
3   {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
4   {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
5   {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
6   {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
7   {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
8   {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
9   {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
10  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
<ipython-input-65-1f2968efe6e5>:72: FutureWarning: The frame.append method is
  results_df = results_df.append({
```

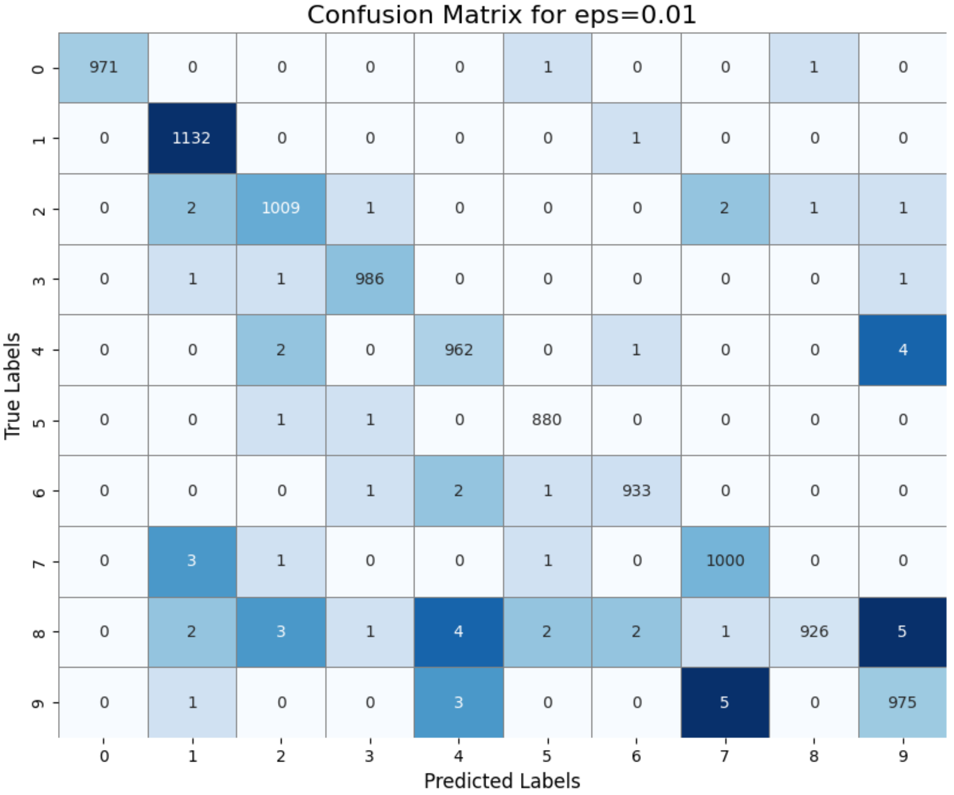
```
# Define the range of eps values
eps_range = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

# Initialize a DataFrame to store results
results_df = pd.DataFrame()

# Loop over the eps values
for eps in eps_range:
    for attack_num in range(1, 6):  # Perform attack 5 times
        # Define the attack with the current eps
        attack = FastGradientMethod(classifier, eps=eps)

        # Apply the attack to generate adversarial examples
        x_adv = attack.generate(x=correct_examples)

        # Predict the labels of the adversarial examples
        y_adv = np.argmax(classifier.predict(x_adv), axis=1)

        nb_correct_adv_pred = np.sum(y_adv == correct_labels)

        print(f"Adversarial test data: eps:{eps}")
        print("Correctly classified: {}".format(nb_correct_adv_pred))
        print("Incorrectly classified: {}".format(len(correct_examples)-nb_correct_adv_pred))

        # For clean examples
        y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
        accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
        print(f"accuracy_clean:{accuracy_clean}")
        # Inside the loop for each eps
        # You have already calculated this for adversarial examples:
        # nb_correct_adv_pred = np.sum(y_adv == correct_labels)
        # Now calculate the accuracy for adversarial examples
        accuracy_adv = nb_correct_adv_pred / len(correct_labels)

        print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
        print(f"Adversarial test data: eps:{eps}")
        print(f"Accuracy on adversarial examples: {accuracy_adv * 100:.2f}%")


        # Calculate the confusion matrix
        cm = confusion_matrix(correct_labels, y_adv, labels=range(10))
        # Draw and save the confusion matrix
        fig, ax = plt.subplots(figsize=(10, 8))

        # Create a mask for the diagonal elements
        mask = np.eye(len(cm), dtype=bool)
        # Plot the heatmap for off-diagonal elements using the mask
        # Use a professional color palette like 'Blues'
        sns.heatmap(cm, mask=mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecol
        # Plot the heatmap for diagonal elements using the inverse of the mask
        # Use the same color palette for consistency
        sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, lineco
        # Labels, title and ticks
        label_names = [f'{i}' for i in range(10)]
        ax.set_xlabel('Predicted Labels', fontsize=12)
        ax.set_ylabel('True Labels', fontsize=12)
        ax.set_title(f'Confusion Matrix for eps={eps}', fontsize=16)
        ax.set_xticklabels(label_names)
        ax.set_yticklabels(label_names)
        image_filename = f'confusion_matrix_eps_{eps}_attack_{attack_num}.png'
        plt.savefig(image_filename, bbox_inches='tight')
        plt.show()  # Close the figure to avoid displaying it in the notebook

        # Save the results in the DataFrame
        results_df = results_df.append({
            'eps': eps,
            'attack_num': attack_num,
            'total_correct': len(correct_labels),
            'total_adv': len(y_adv),
            'correct_adv_counts': dict(zip(*np.unique(y_adv, return_counts=True)))
        }, ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('/content/drive/MyDrive/ColabNotebooks/5attack_withouttruelabel_adv_results.csv', index=False

# Print the DataFrame
print(results_df)
```

```
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

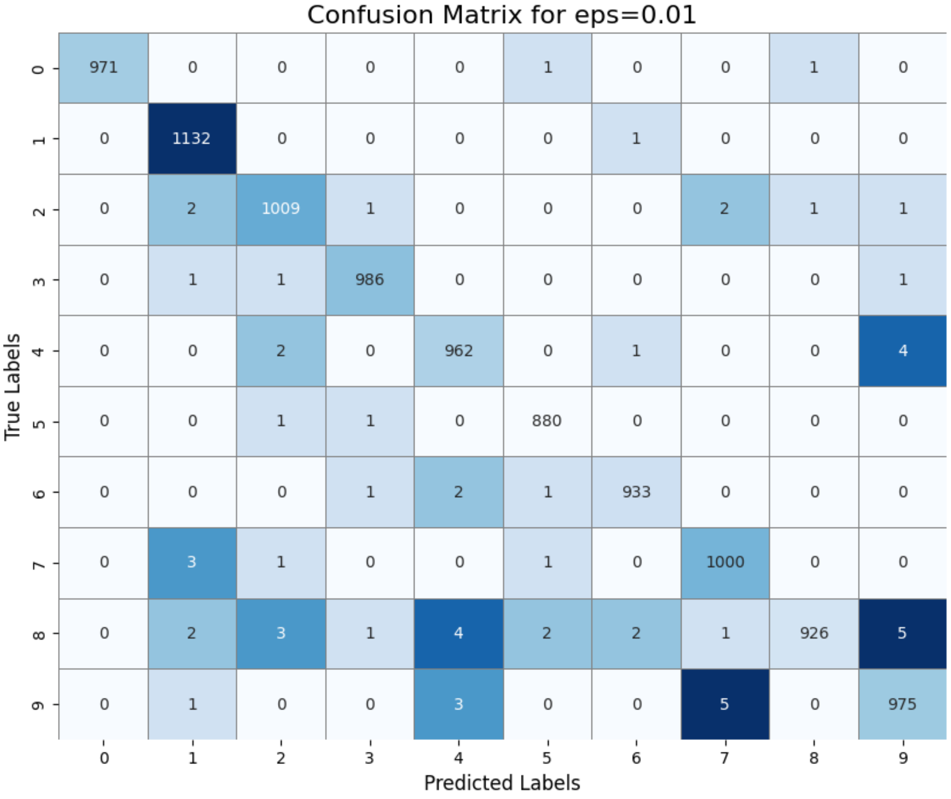
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

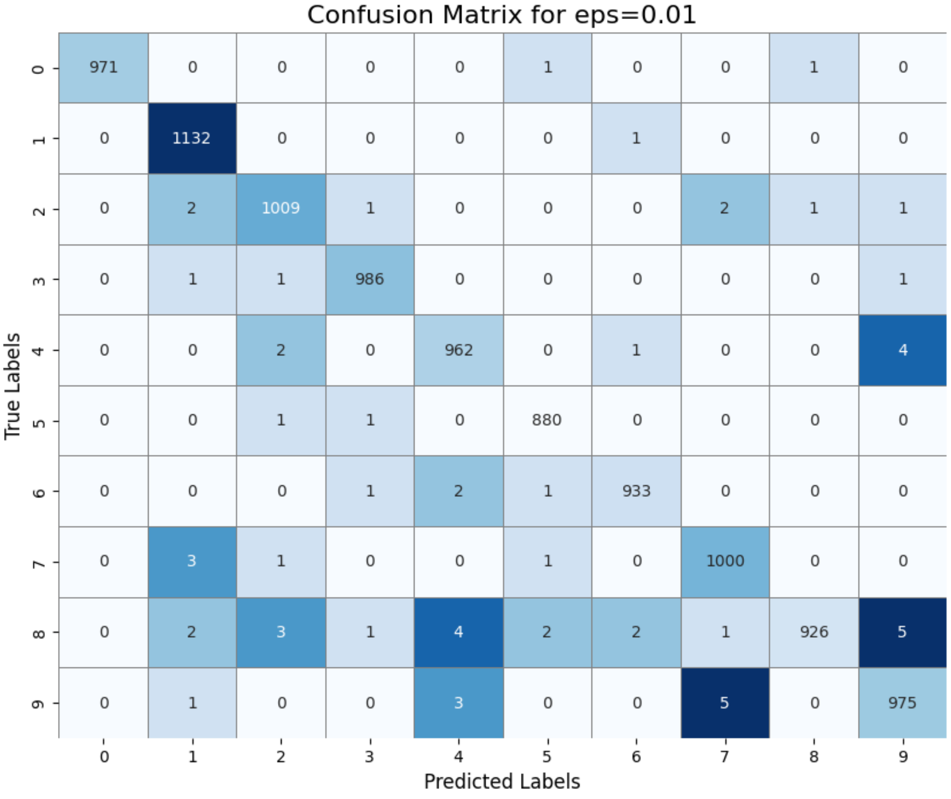
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
```

```
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



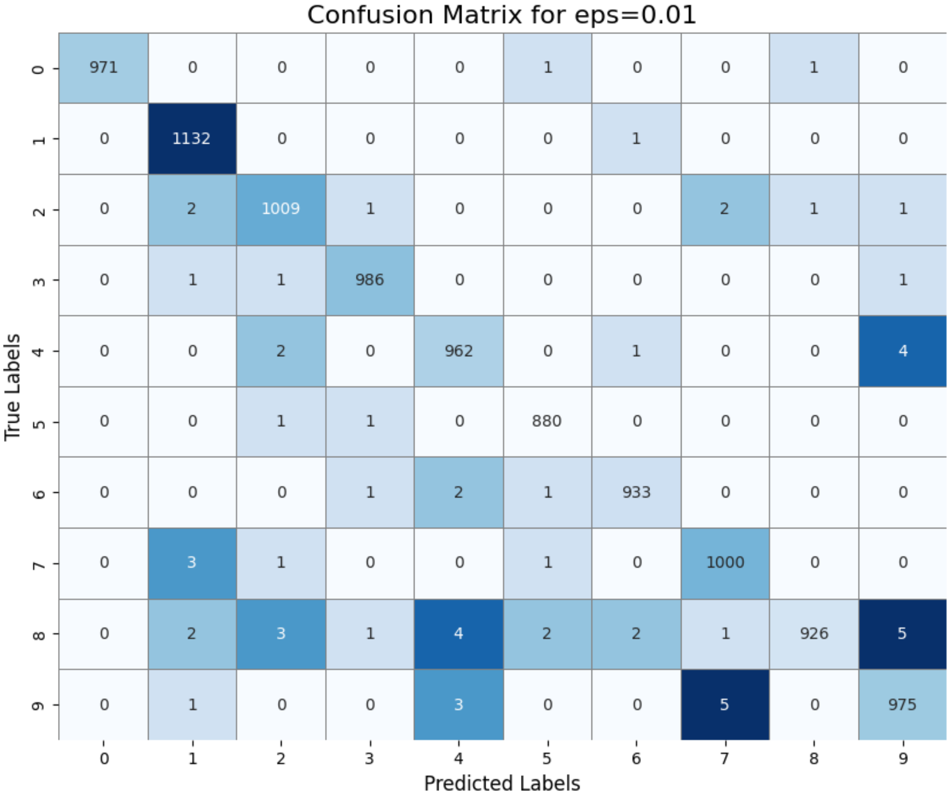
	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



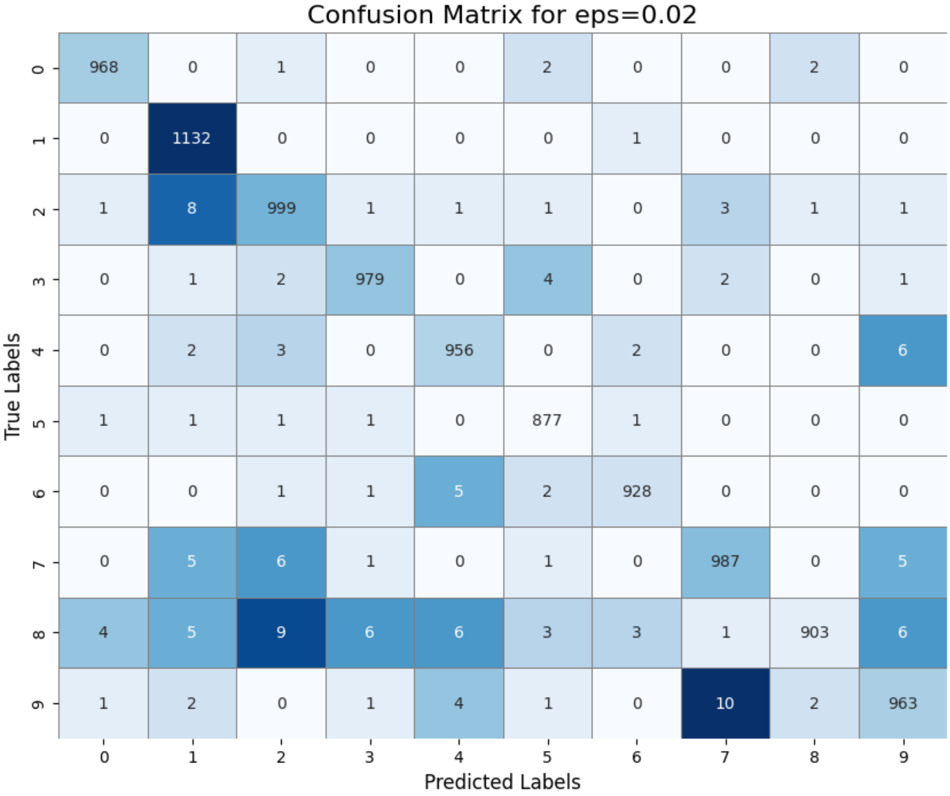
	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

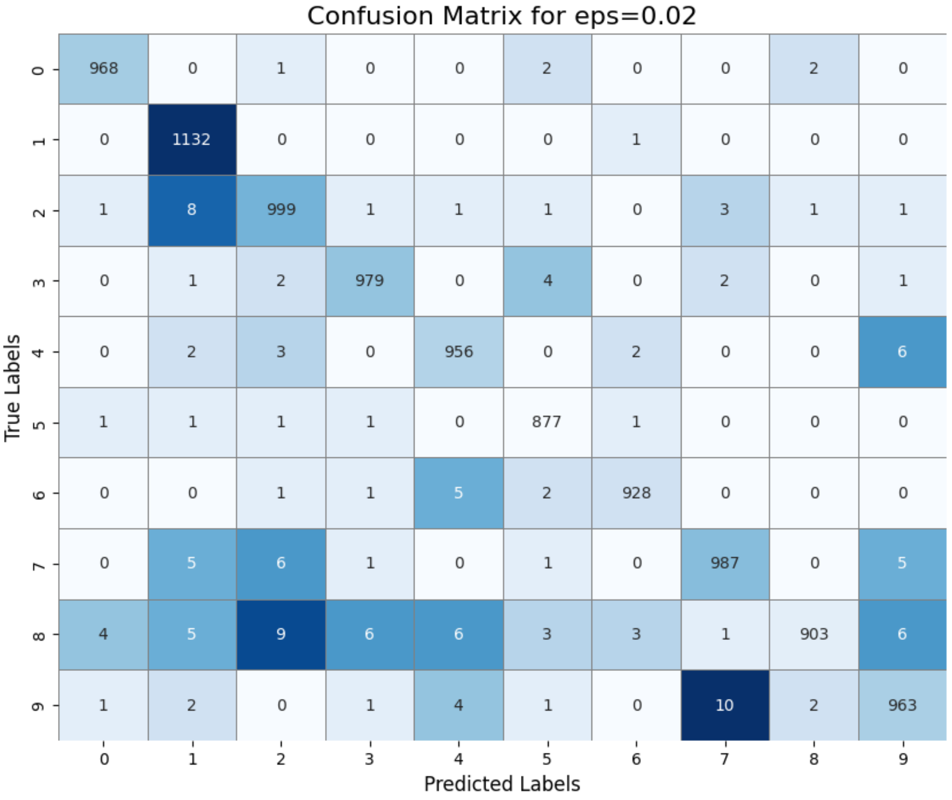
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



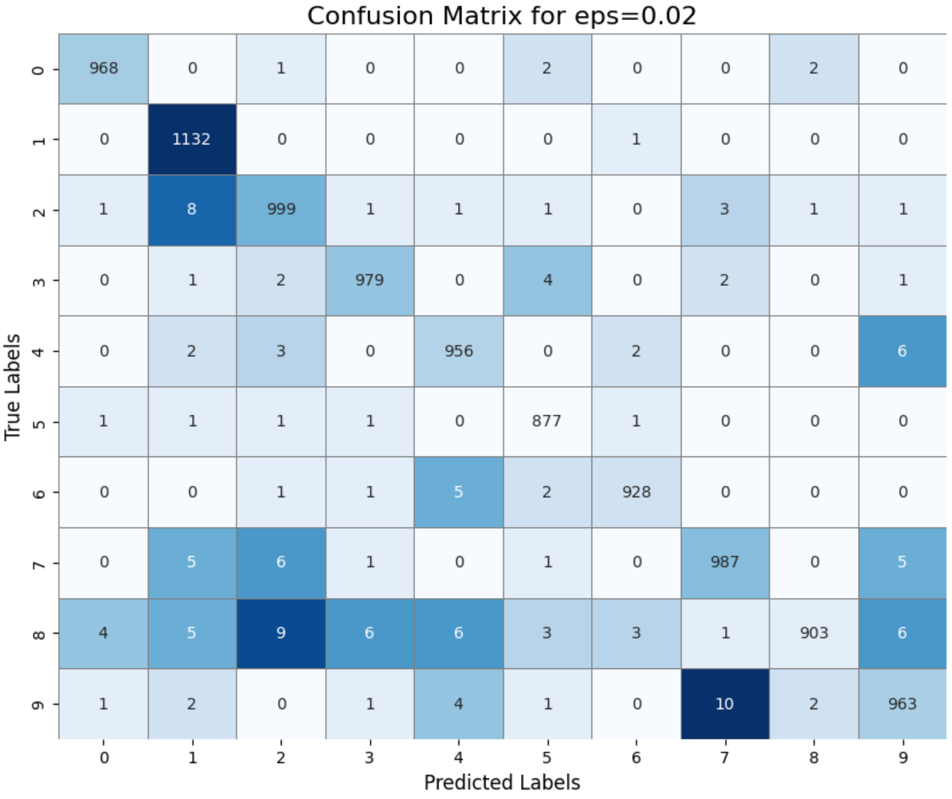
	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
```

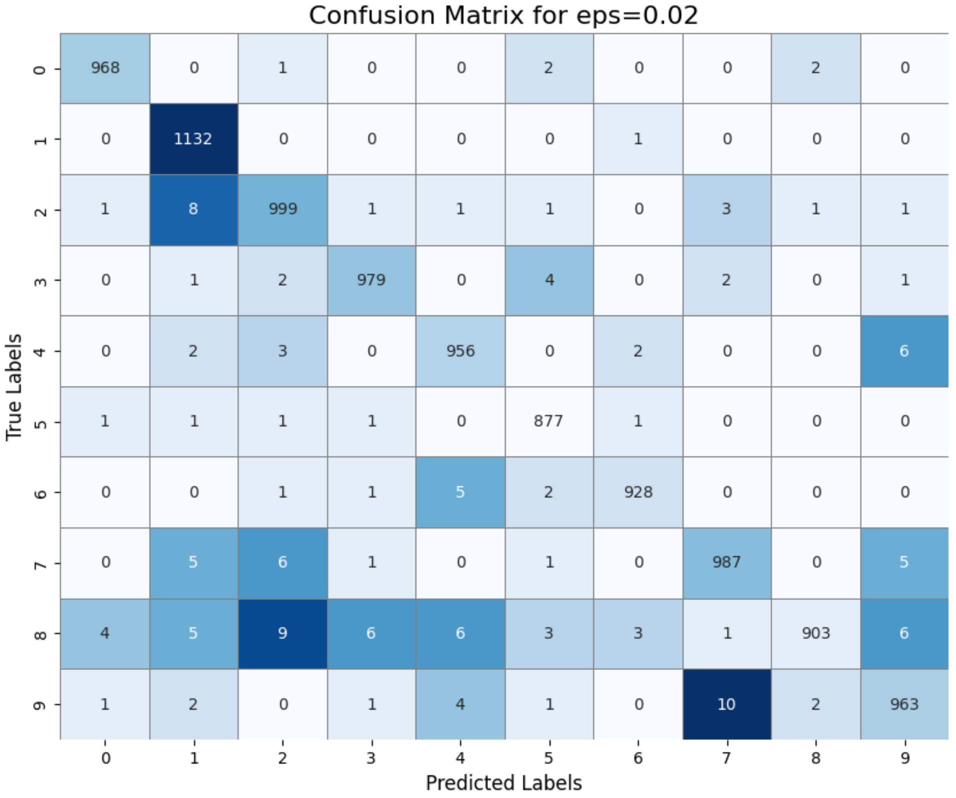
```
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



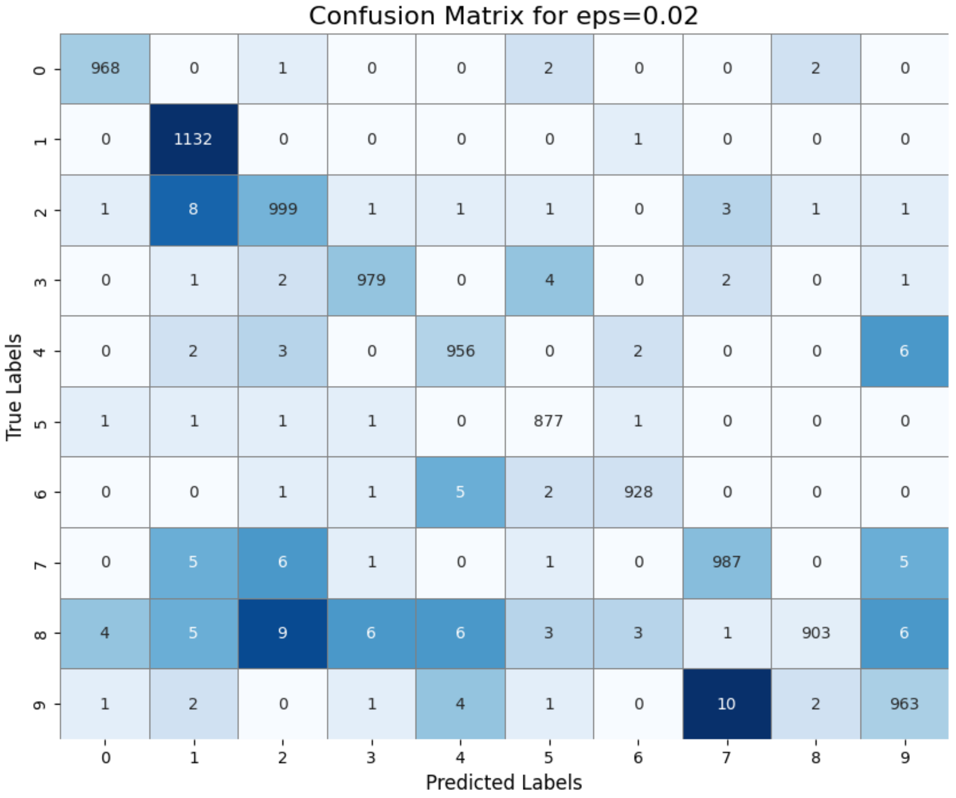
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```

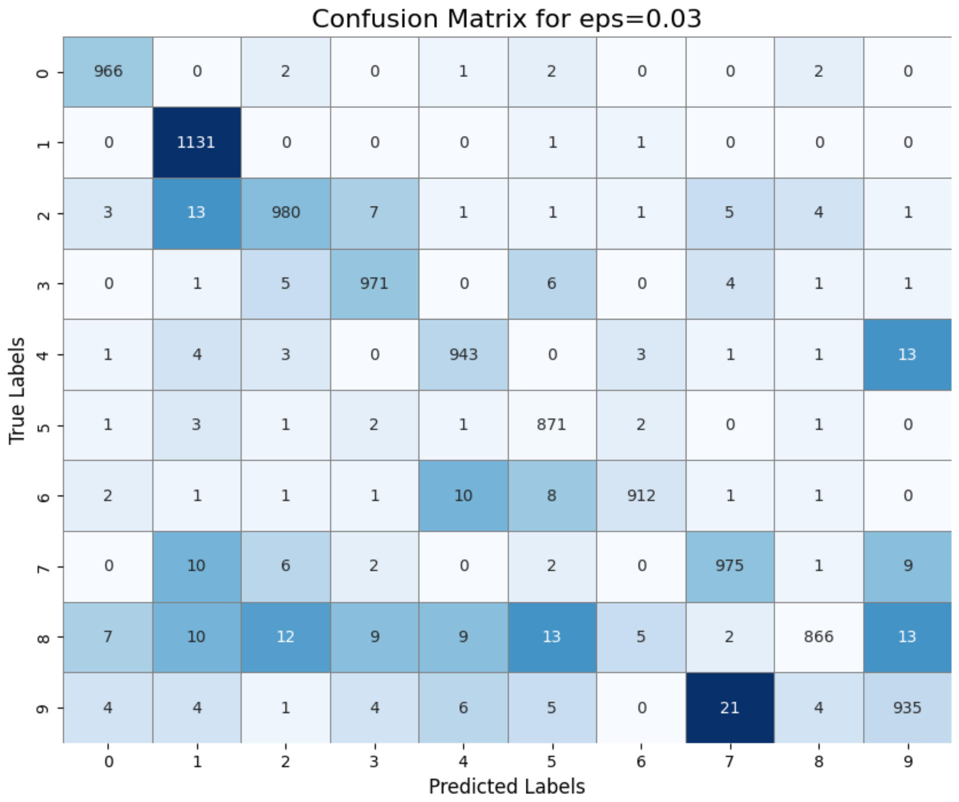


```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```

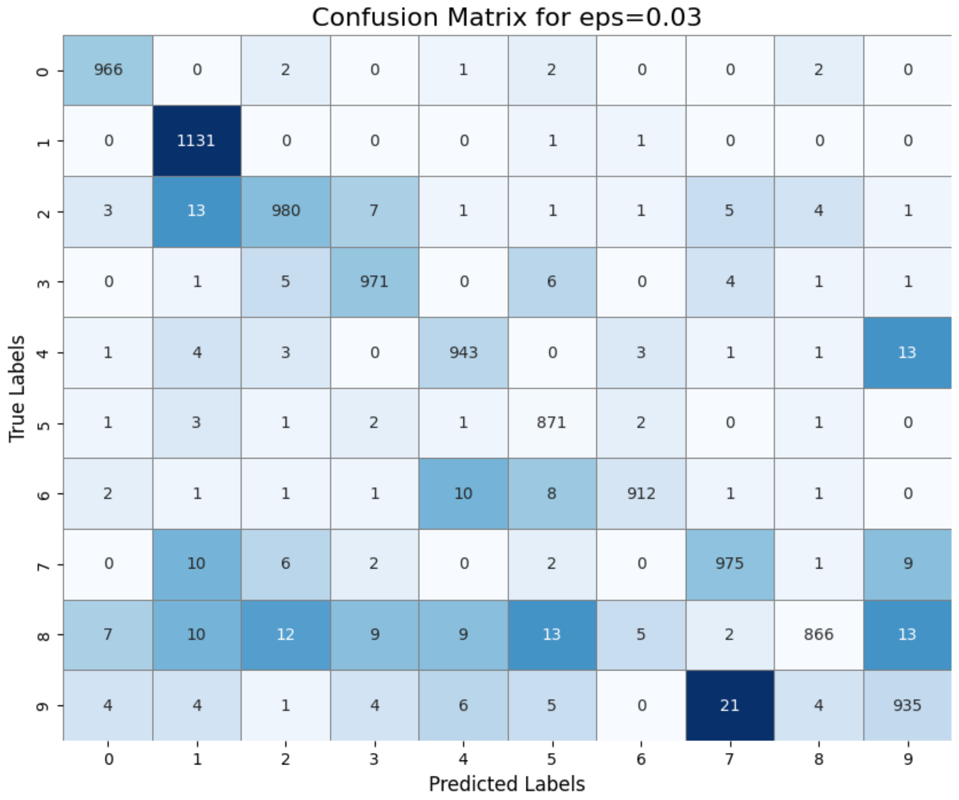


```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
```

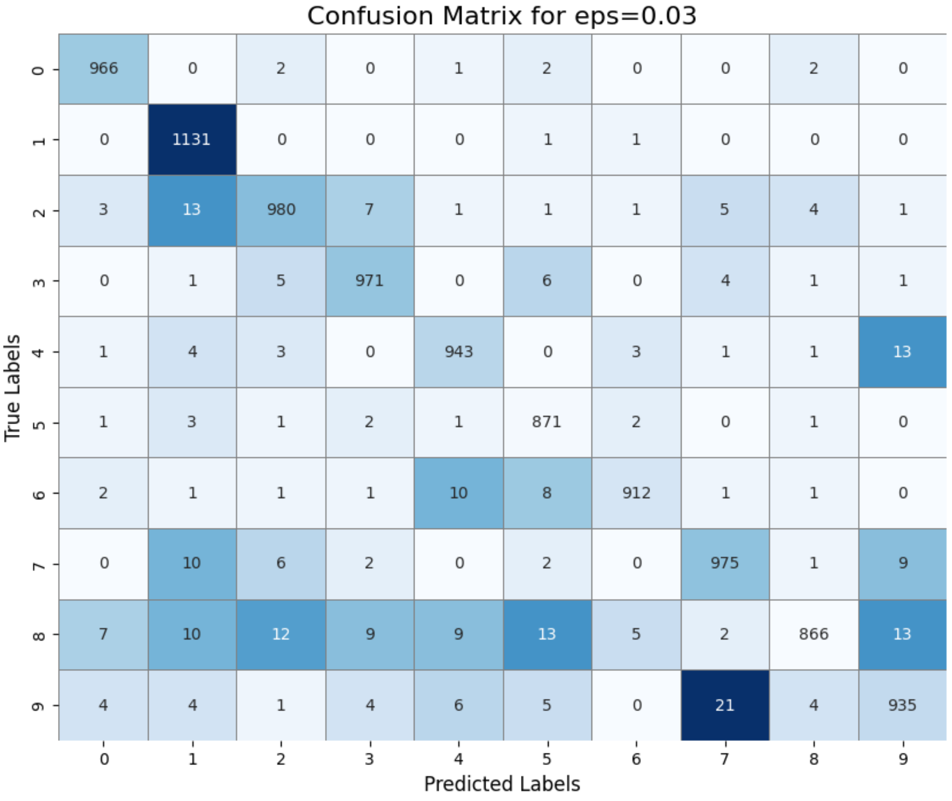
```
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



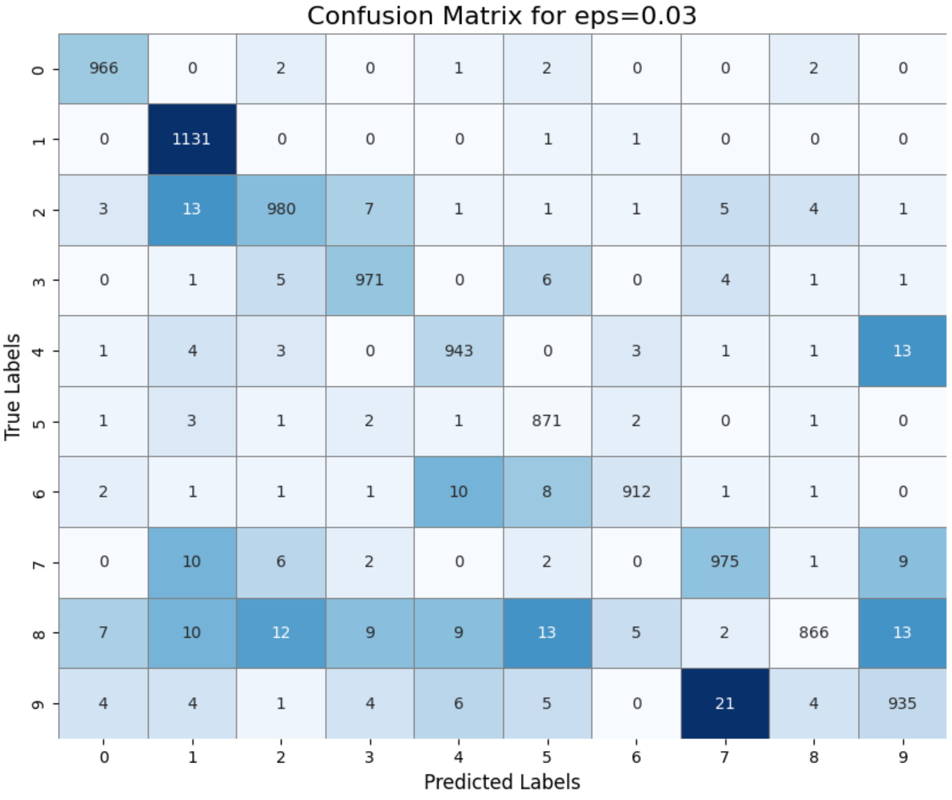
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



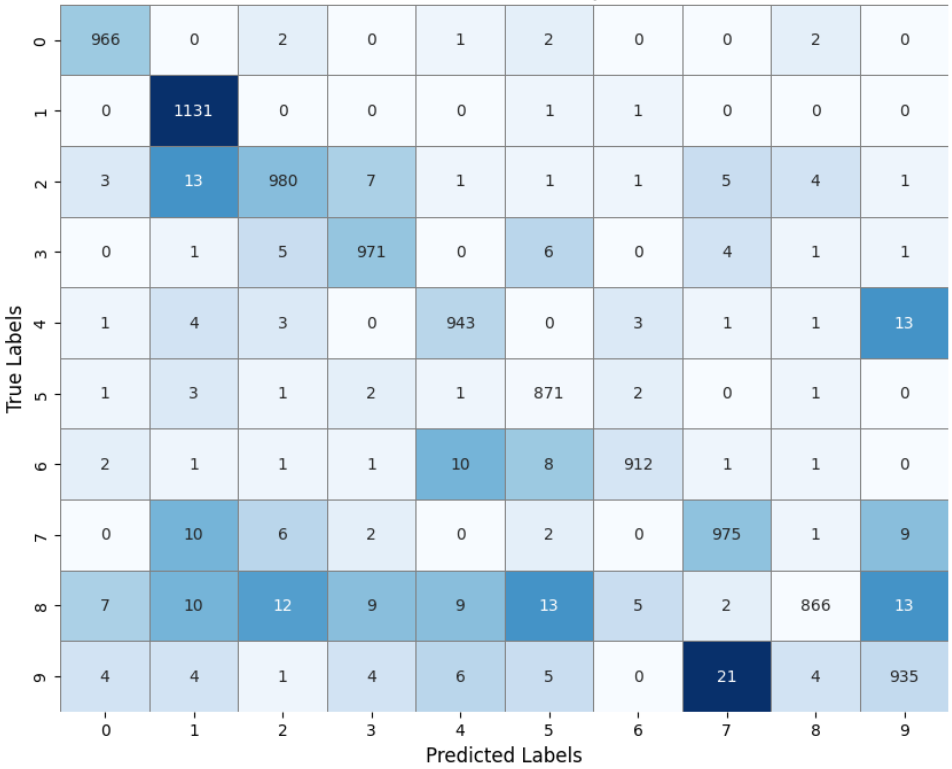
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
```

```
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```

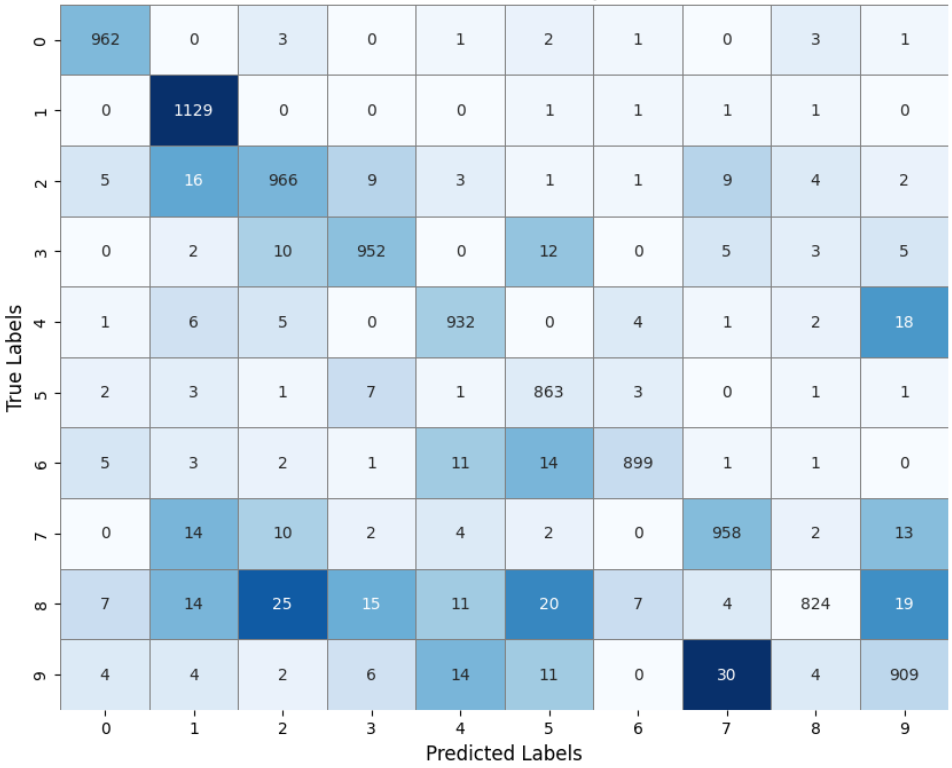
Confusion Matrix for eps=0.03



	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

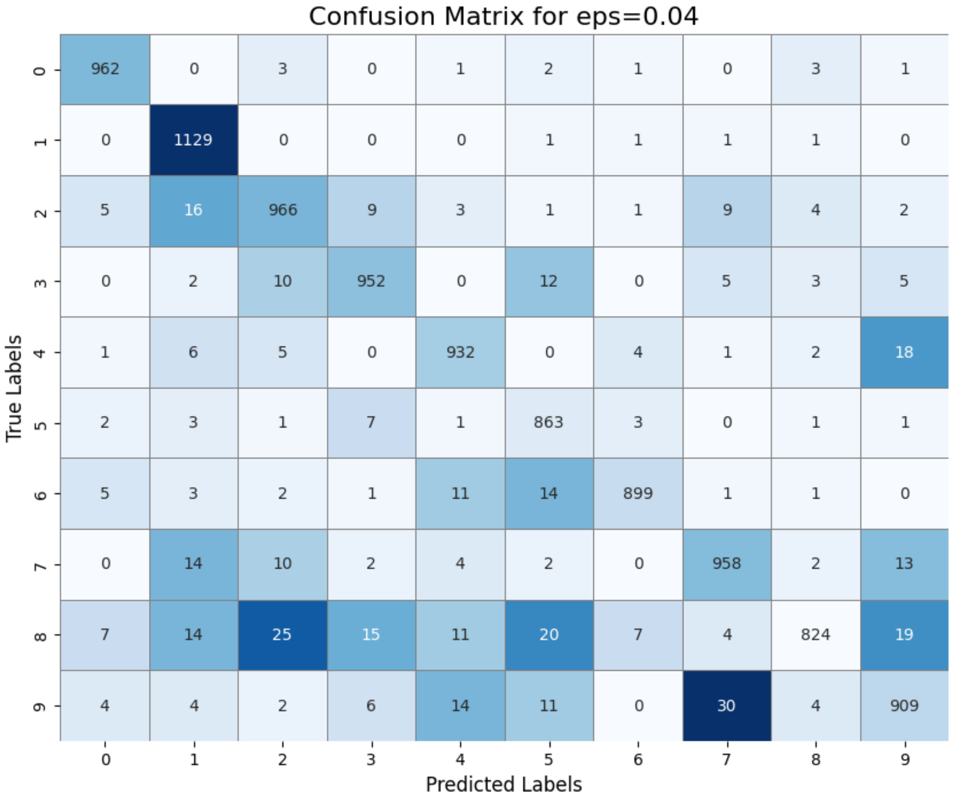
Confusion Matrix for eps=0.04



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

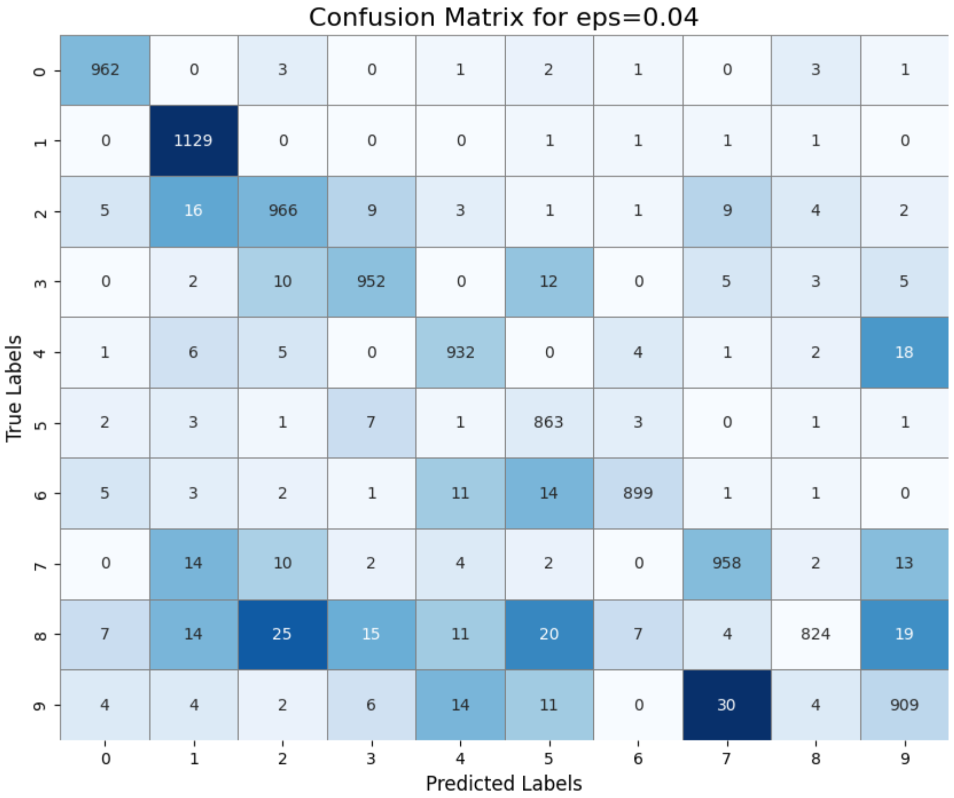
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
```

```
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

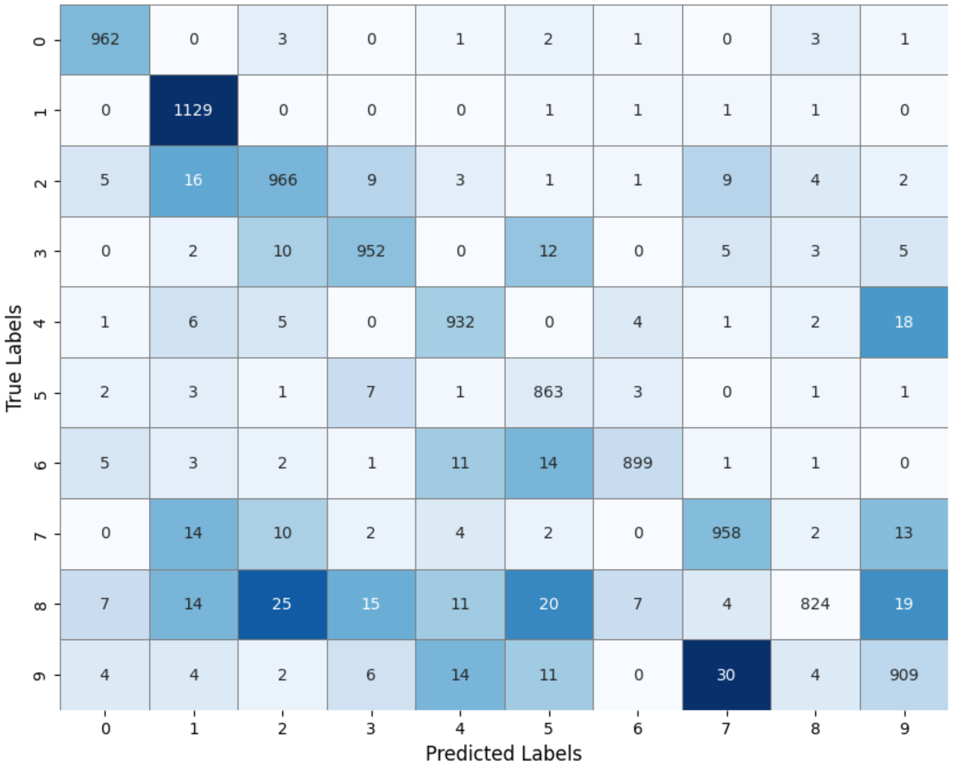


	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
```

```
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

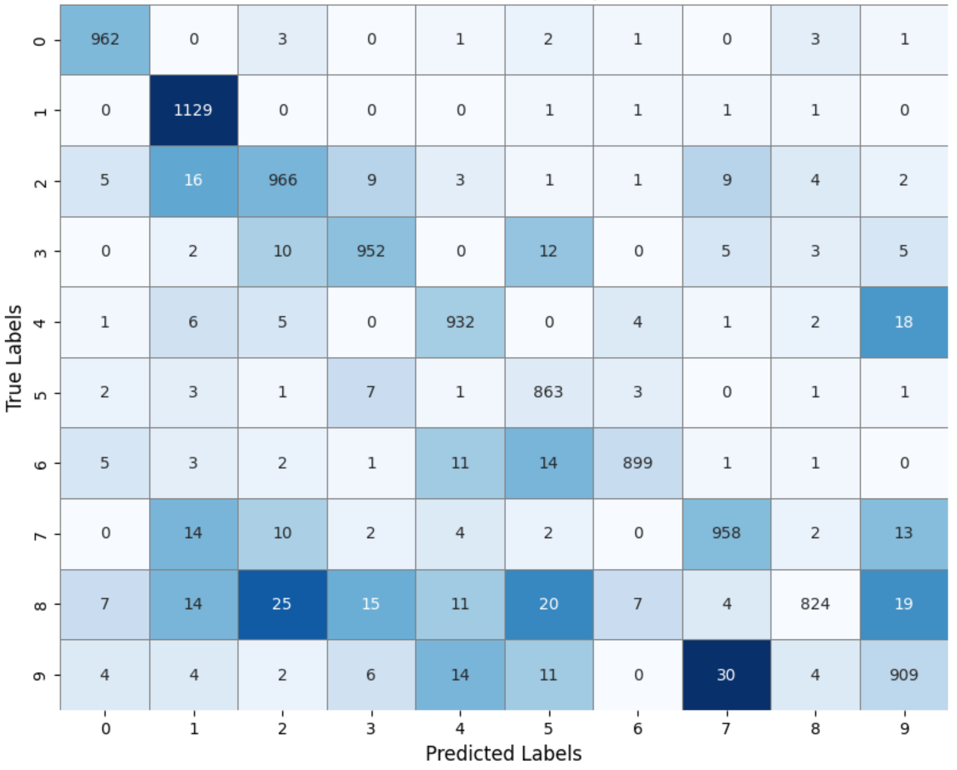
Confusion Matrix for eps=0.04



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

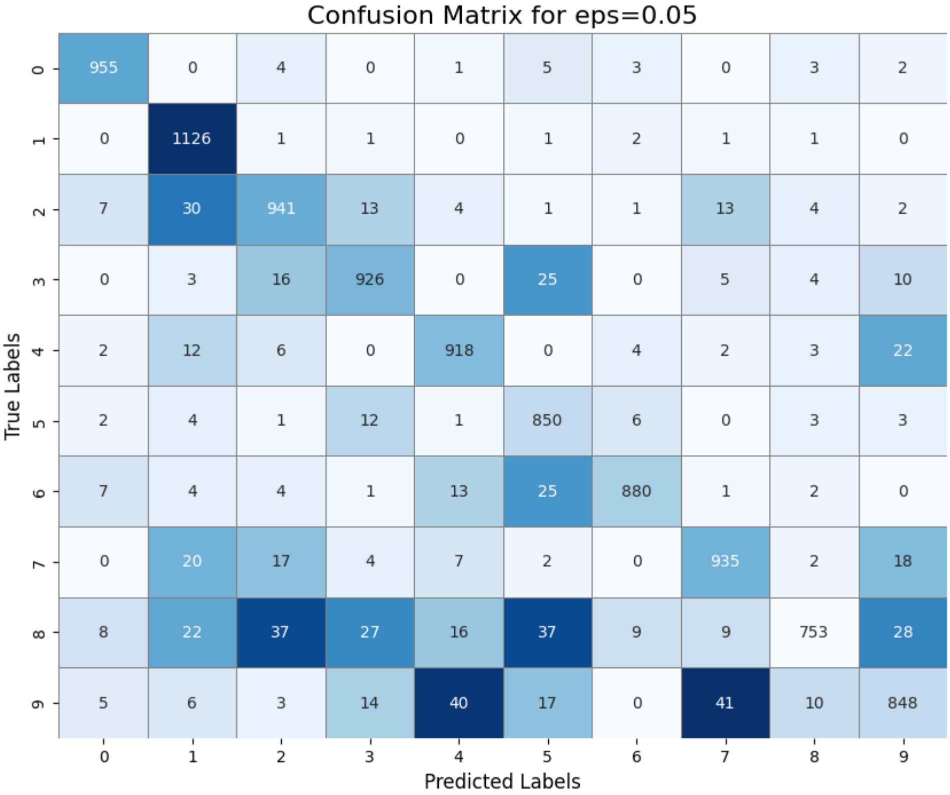
Confusion Matrix for eps=0.04



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

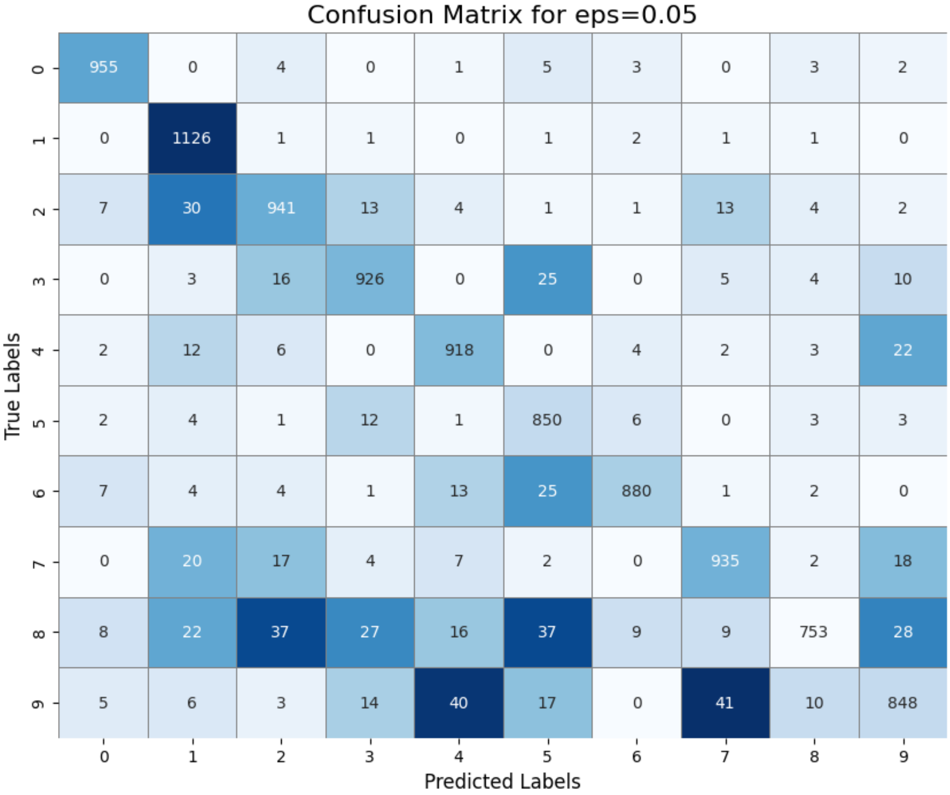
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
```

```
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```


Confusion Matrix for eps=0.05

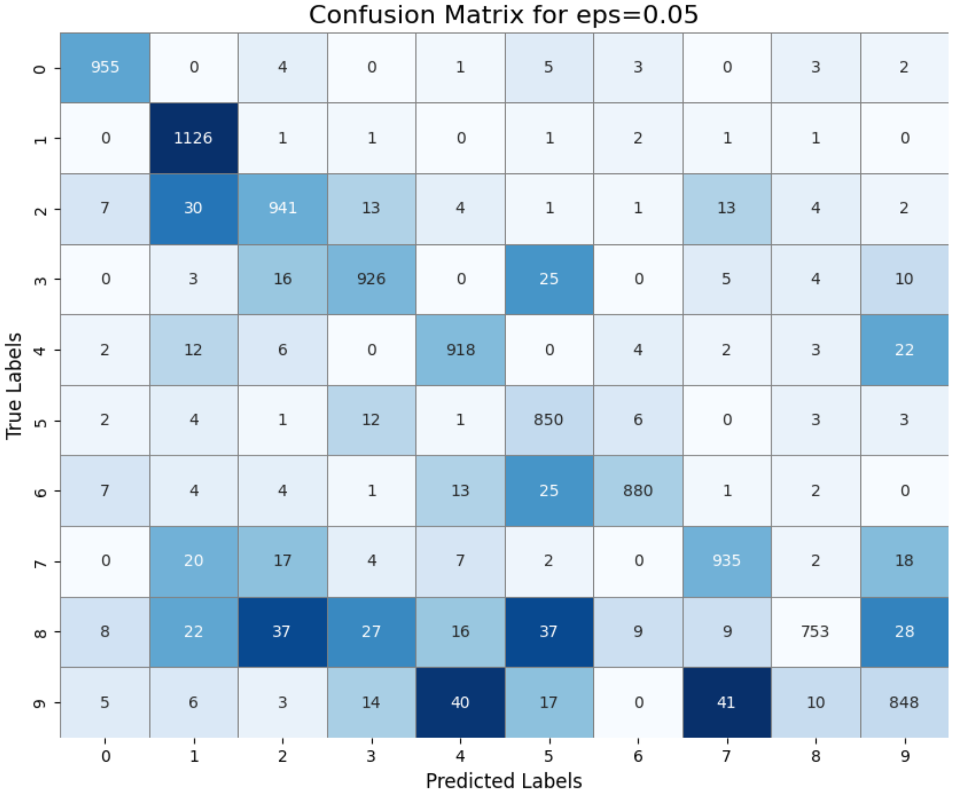
	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```


Confusion Matrix for eps=0.05

	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

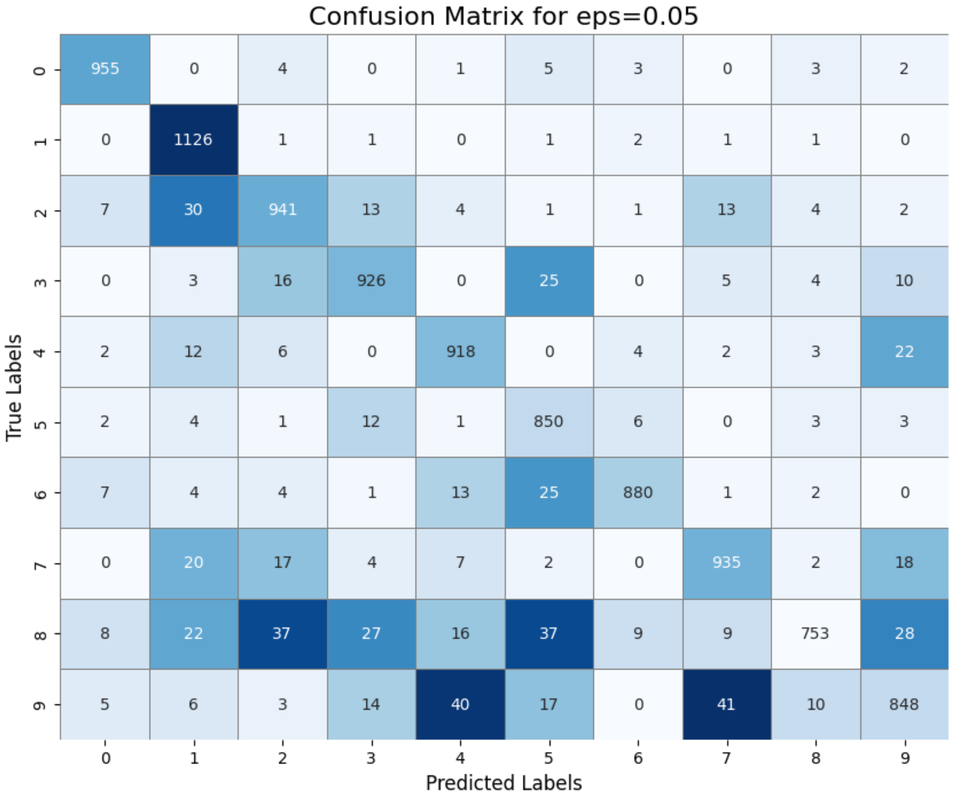
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



Confusion Matrix for eps=0.05

	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



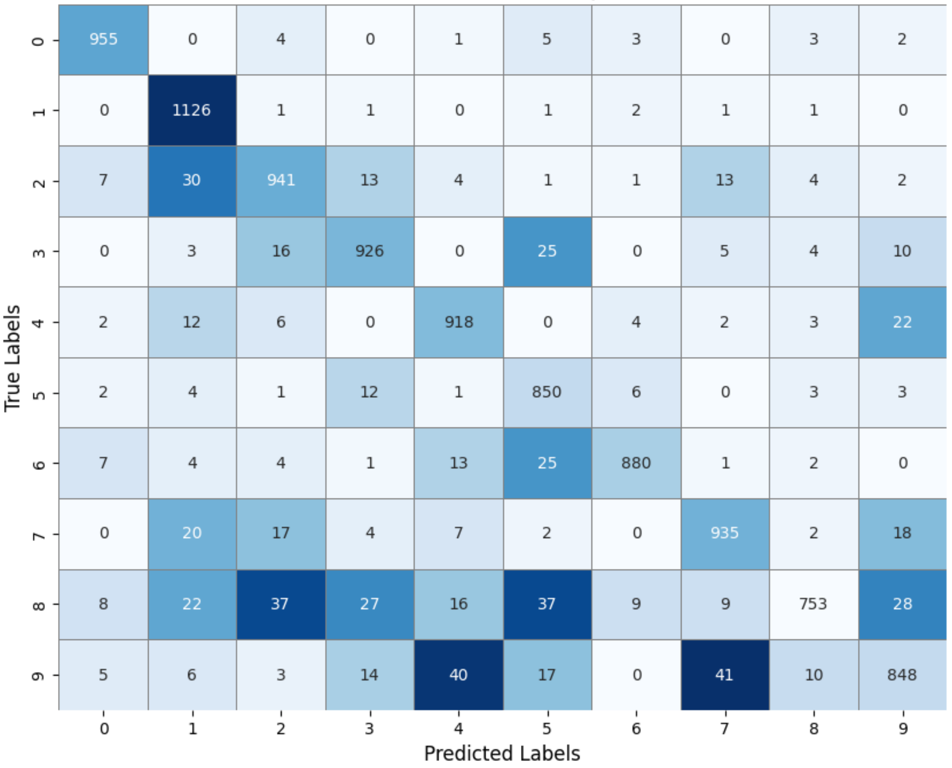
Confusion Matrix for eps=0.05

	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
```

```
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```

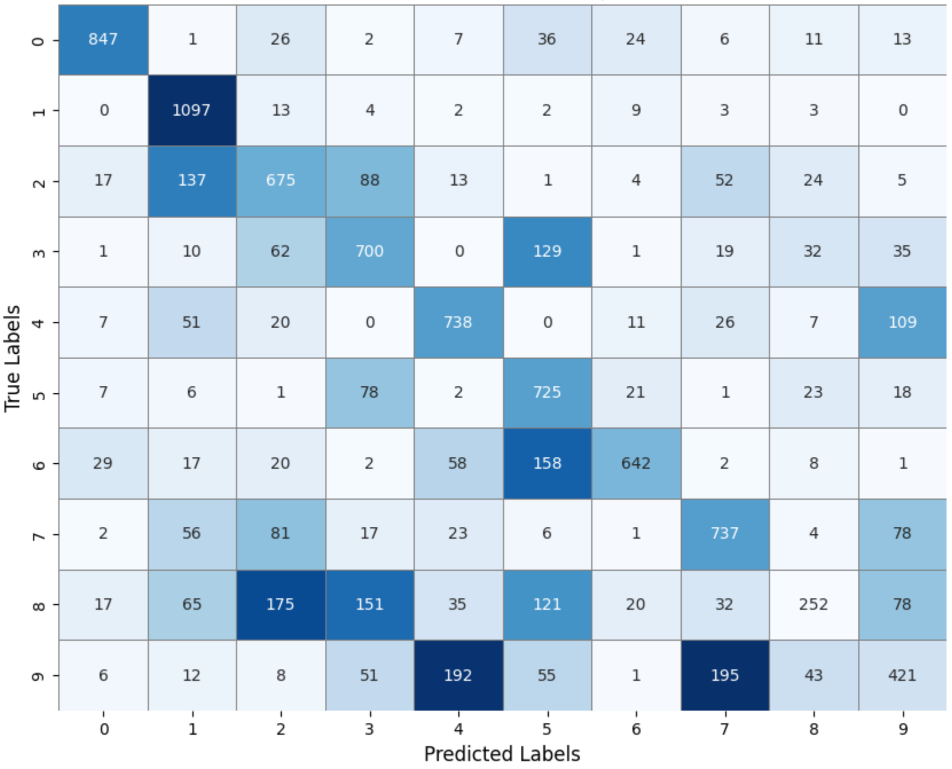
Confusion Matrix for eps=0.05



	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

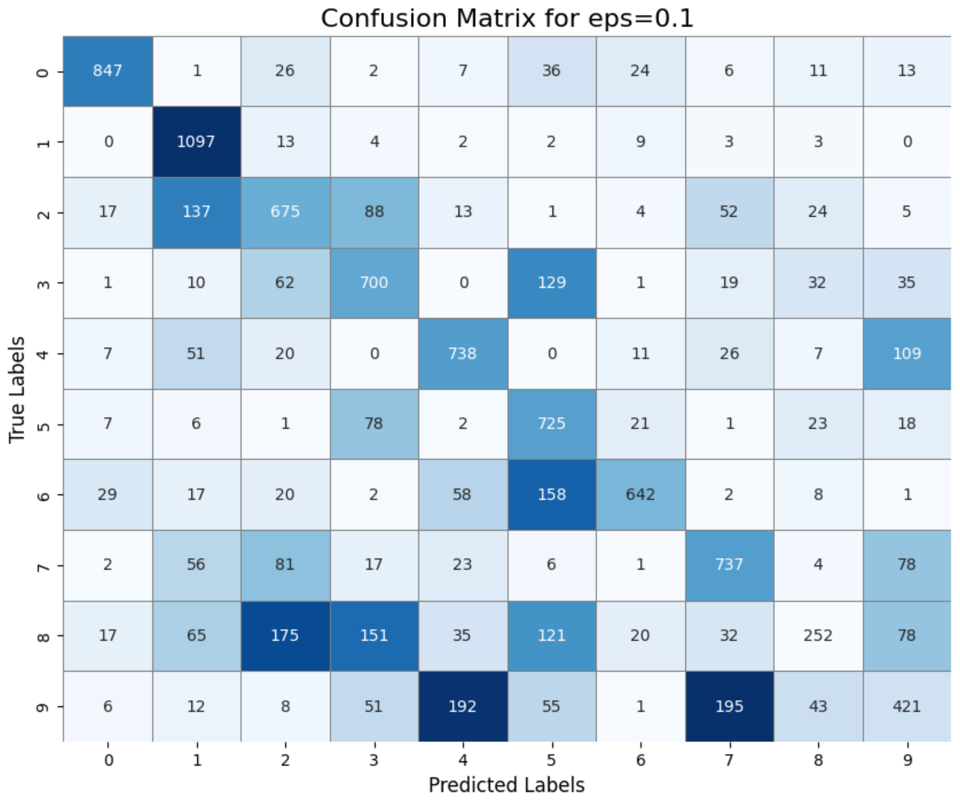
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

Confusion Matrix for eps=0.1



	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

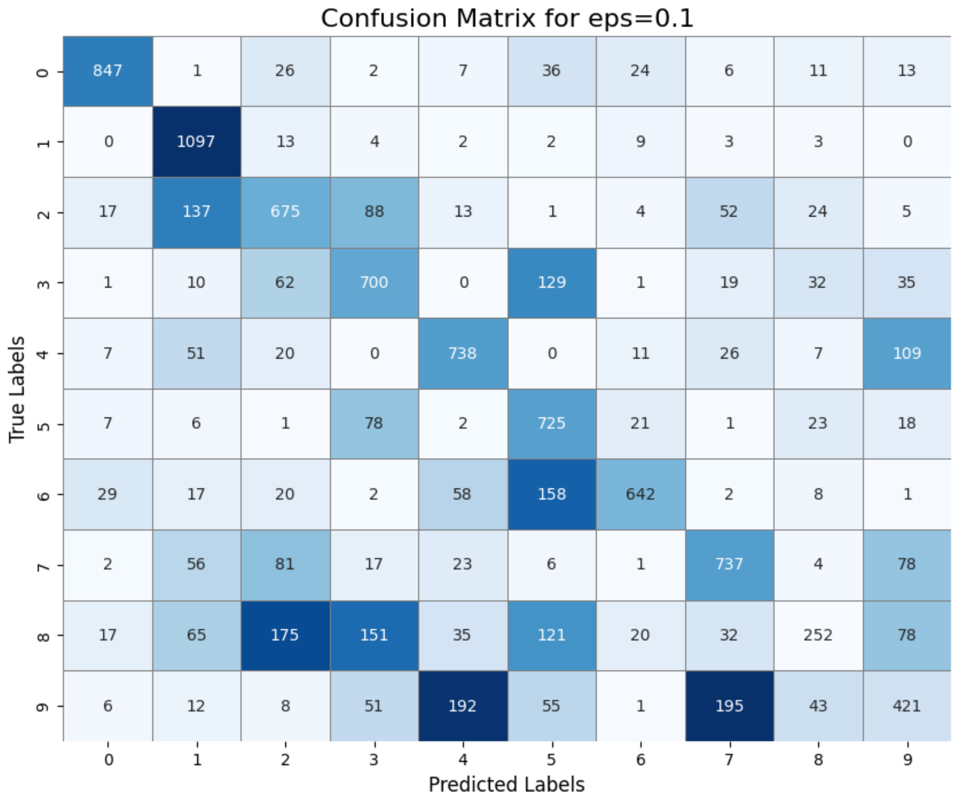
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```



Confusion Matrix for eps=0.1

	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

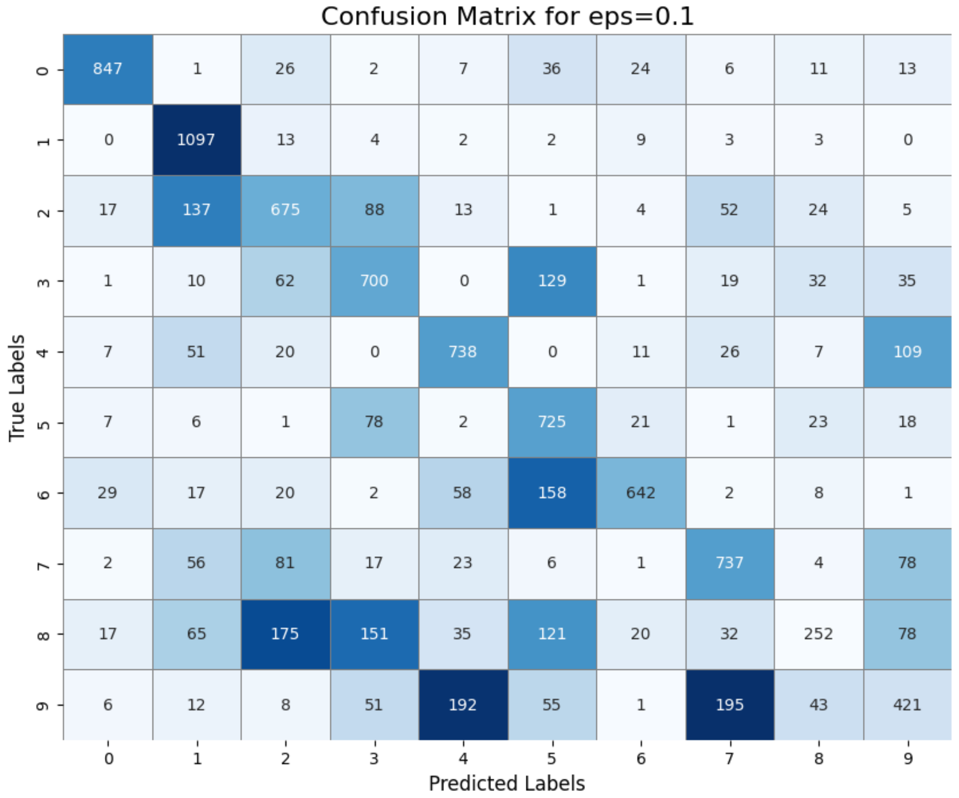


Confusion Matrix for eps=0.1

	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

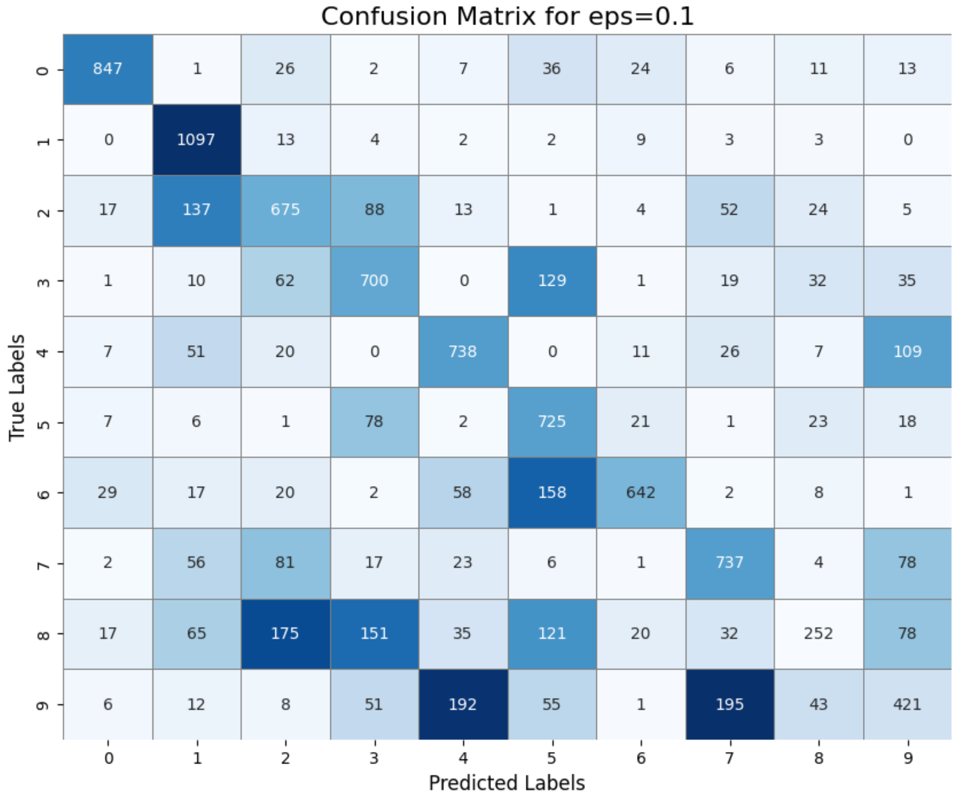
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
```

```
Accuracy on adversarial examples: 69.49%
```



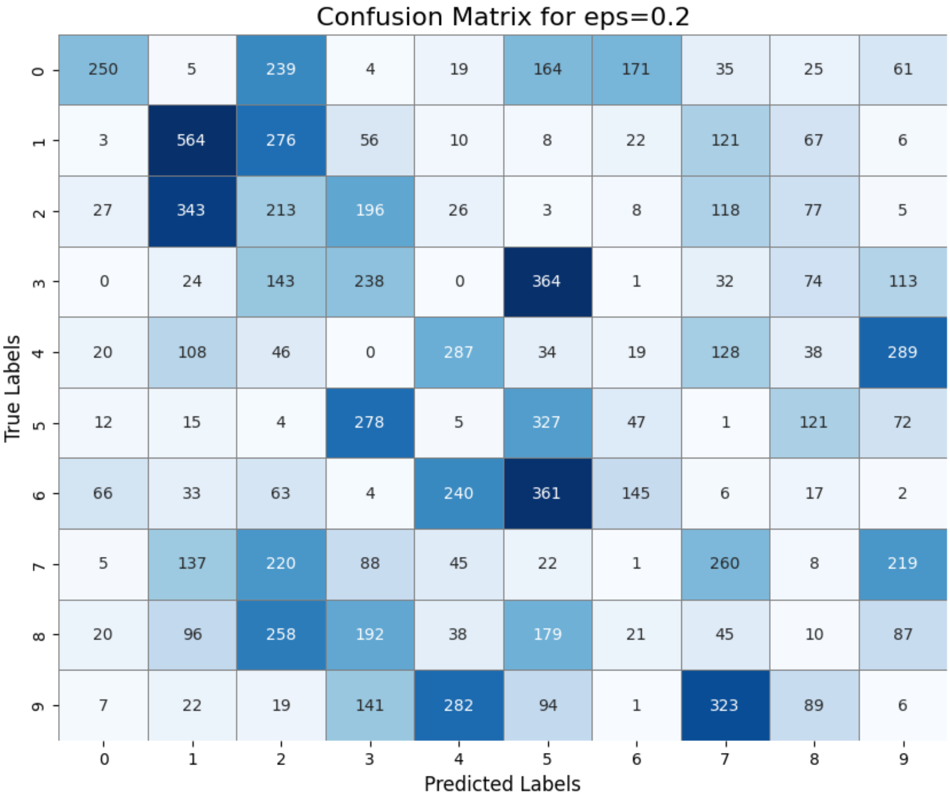
	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```



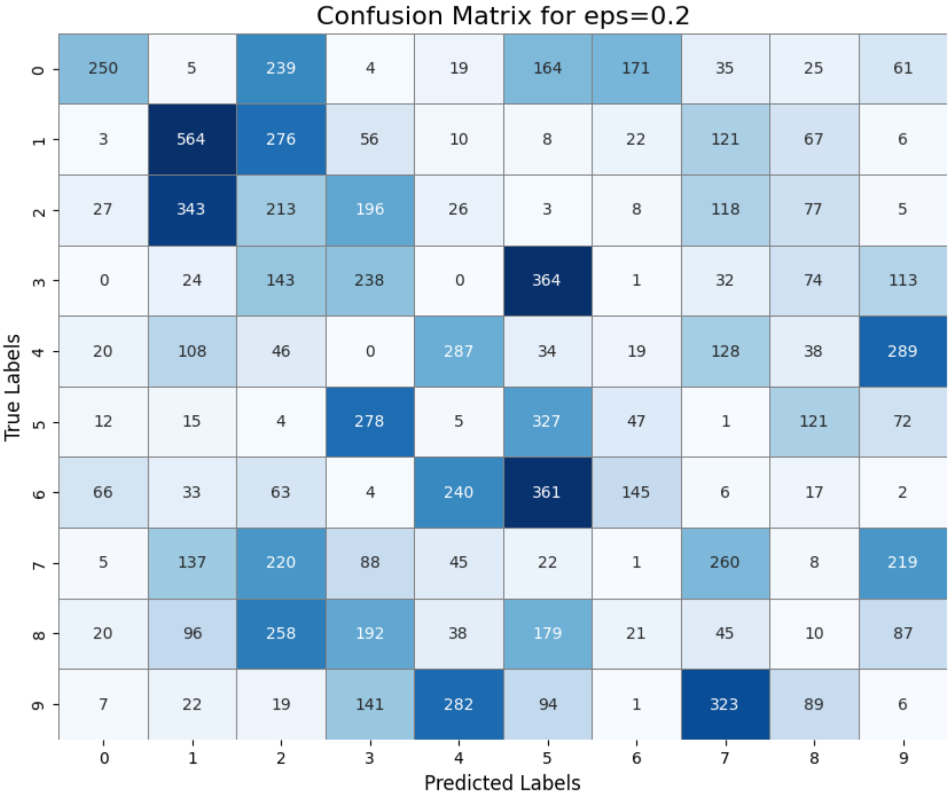
	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
Accuracy on adversarial examples: 23.39%
```



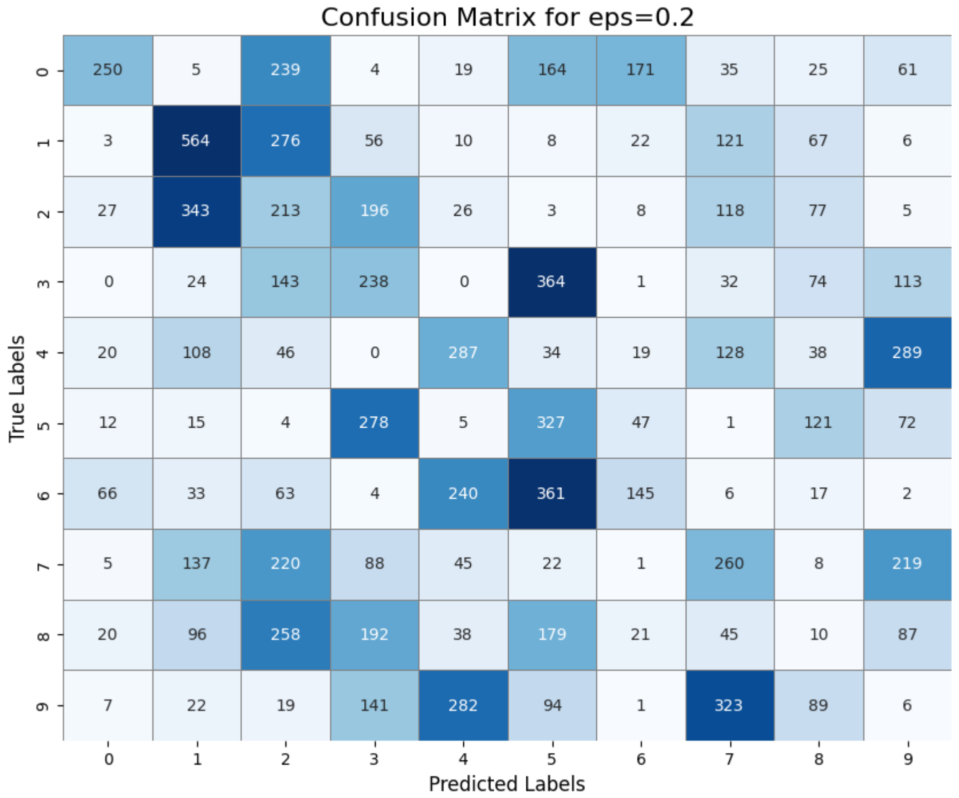
	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

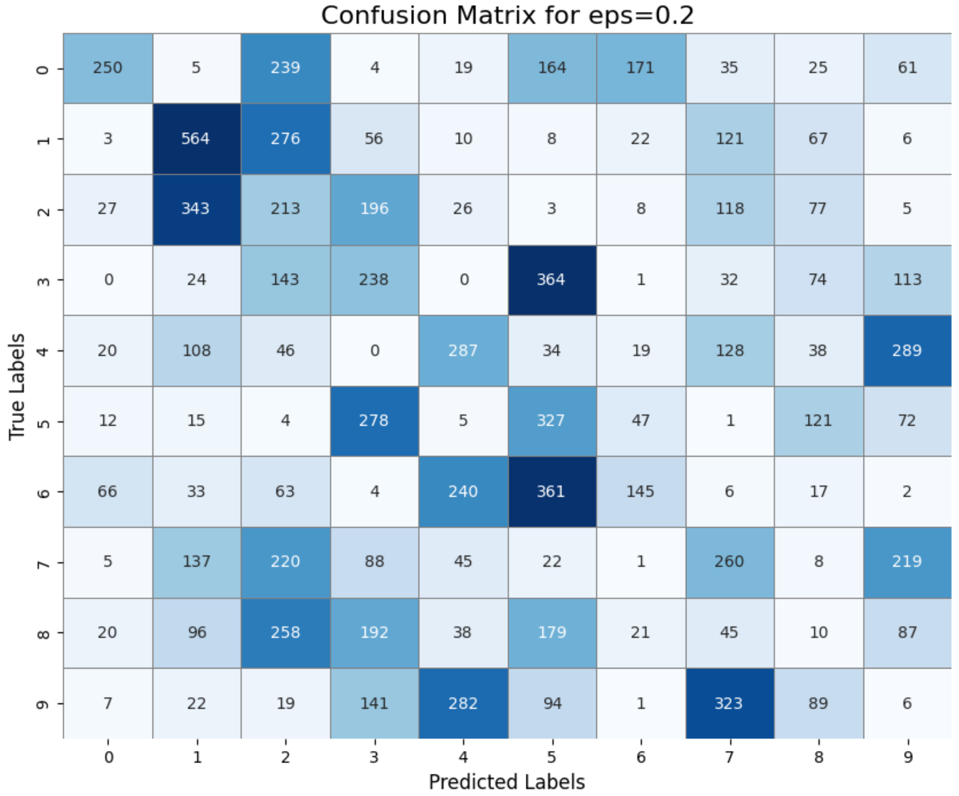
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```



Confusion Matrix for eps=0.2

	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

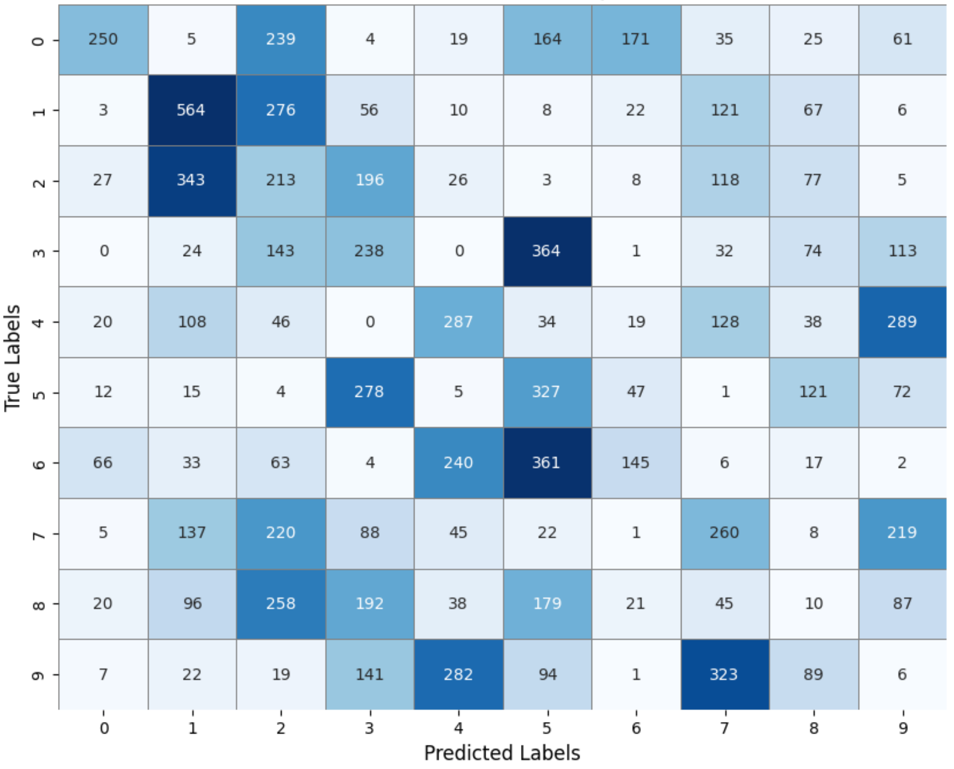


Confusion Matrix for eps=0.2

	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

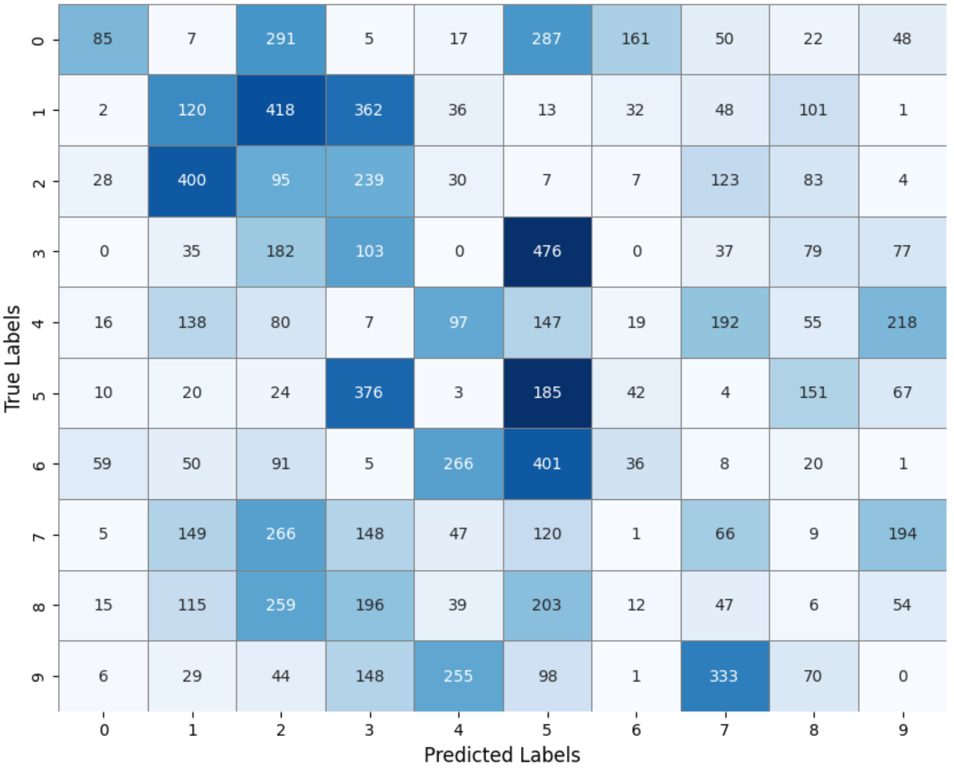
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

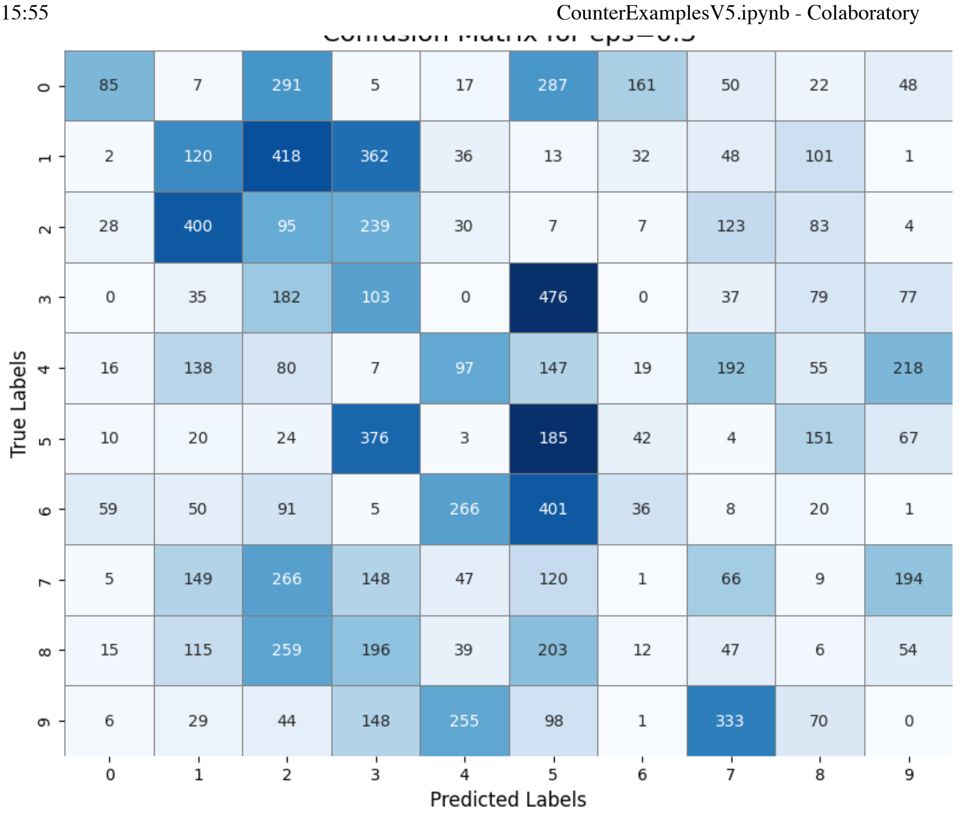
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

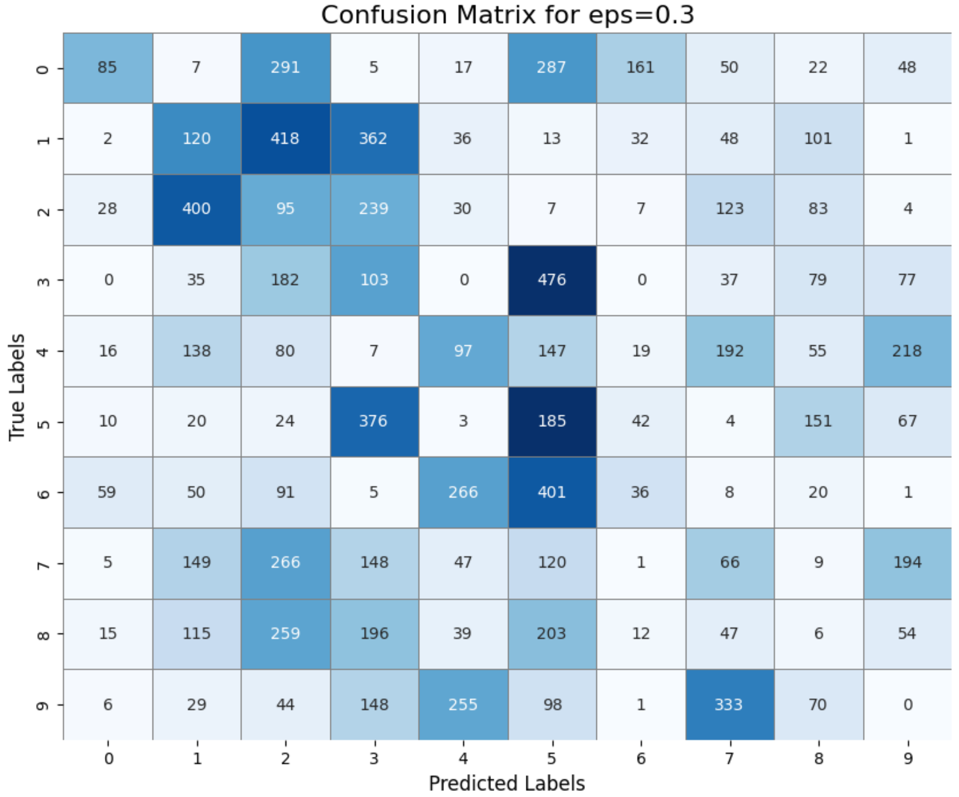
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

Confusion Matrix for eps=0.3



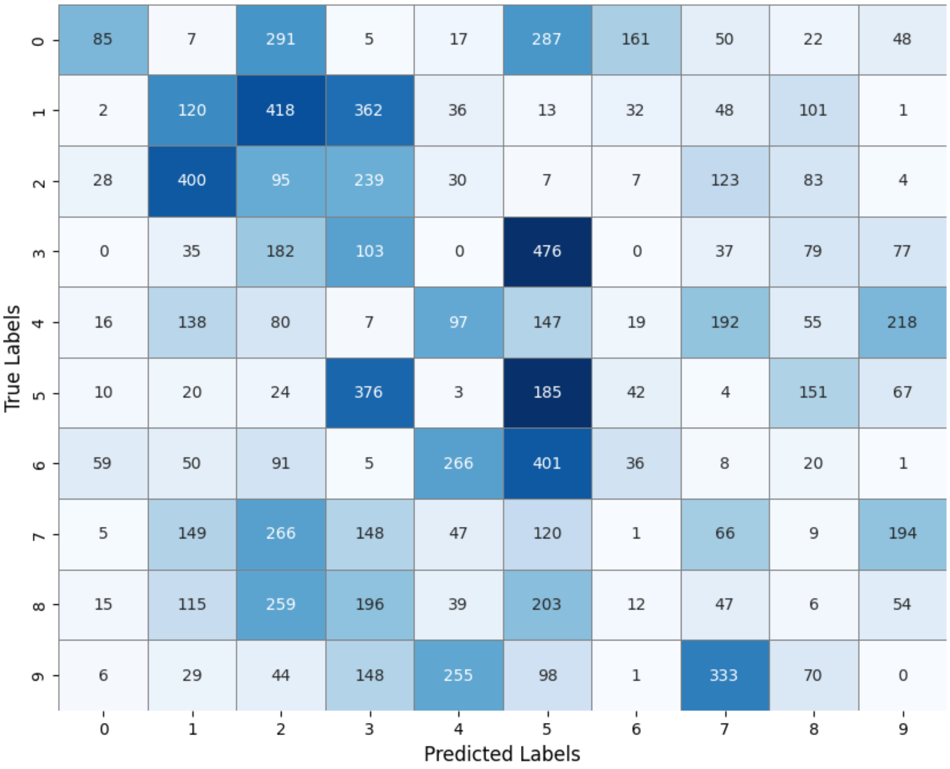
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

Confusion Matrix for eps=0.3



```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

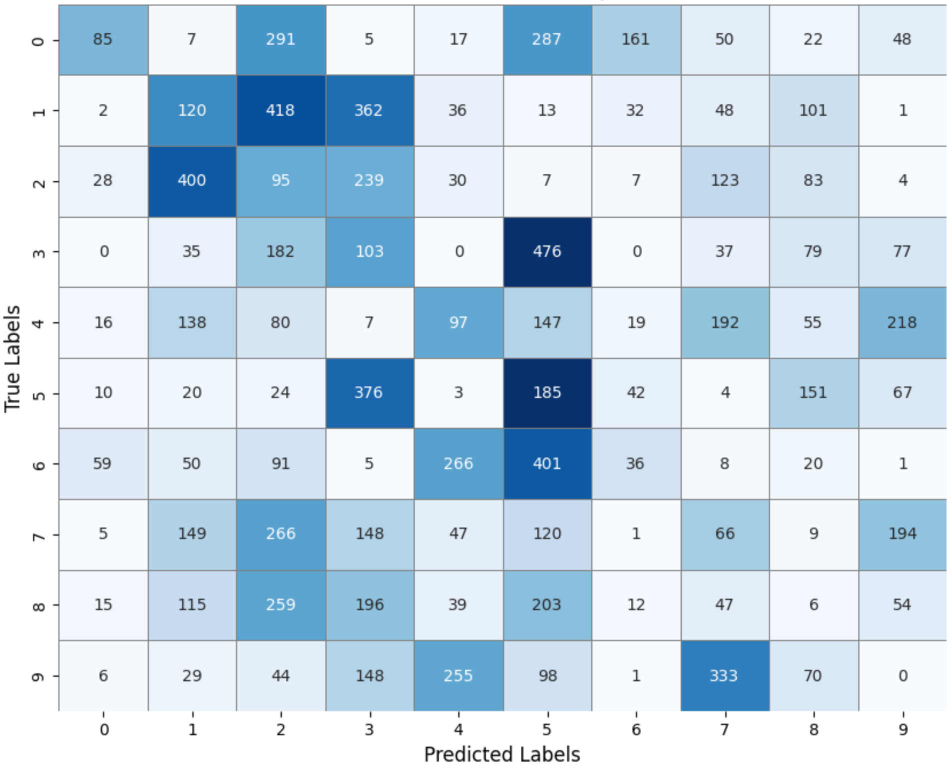
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

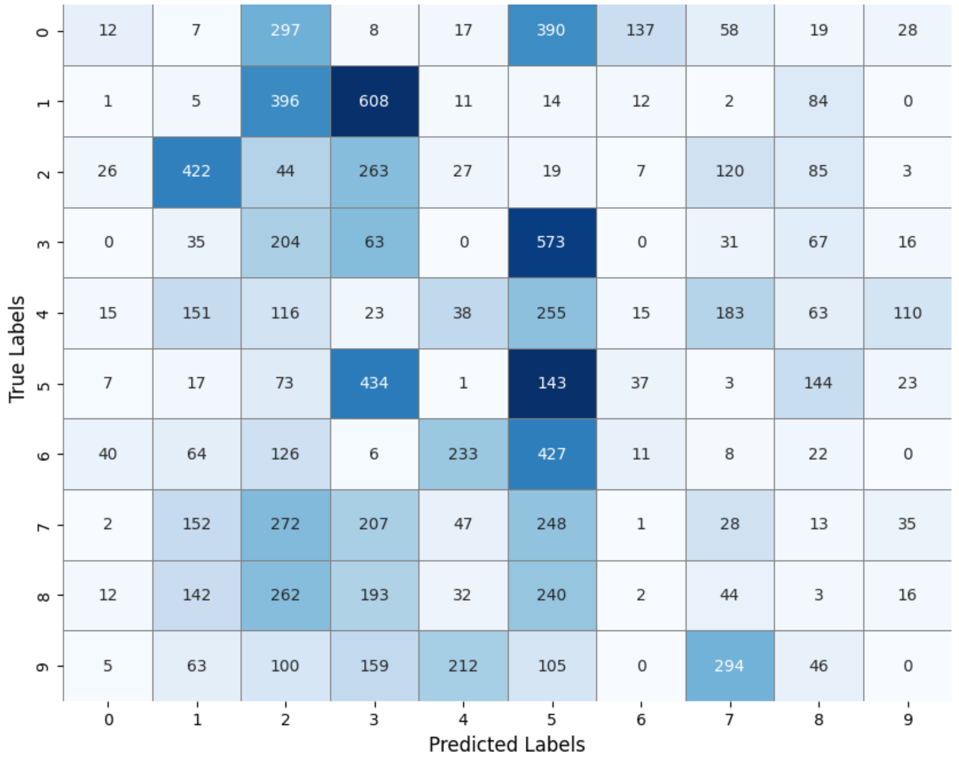
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

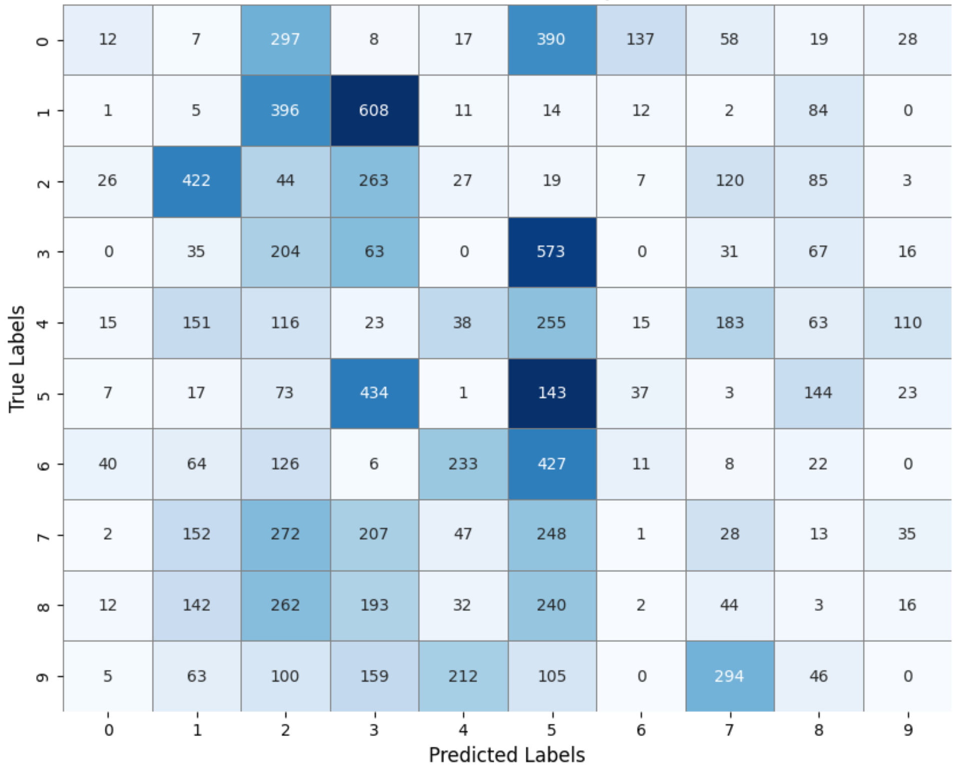
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

Confusion Matrix for eps=0.4



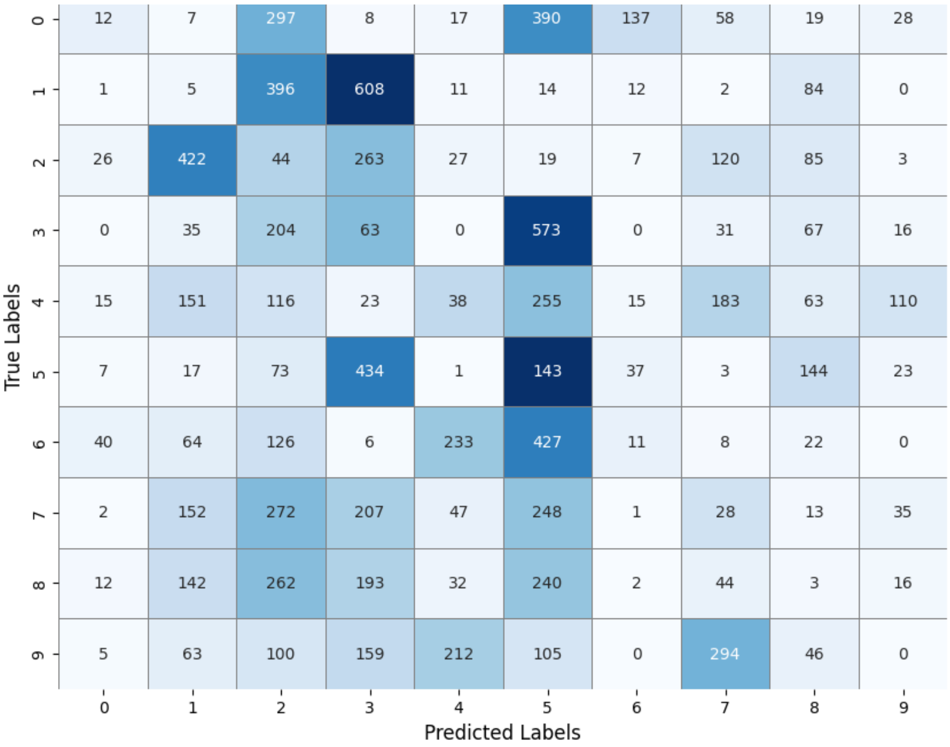
```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

Confusion Matrix for eps=0.4



```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

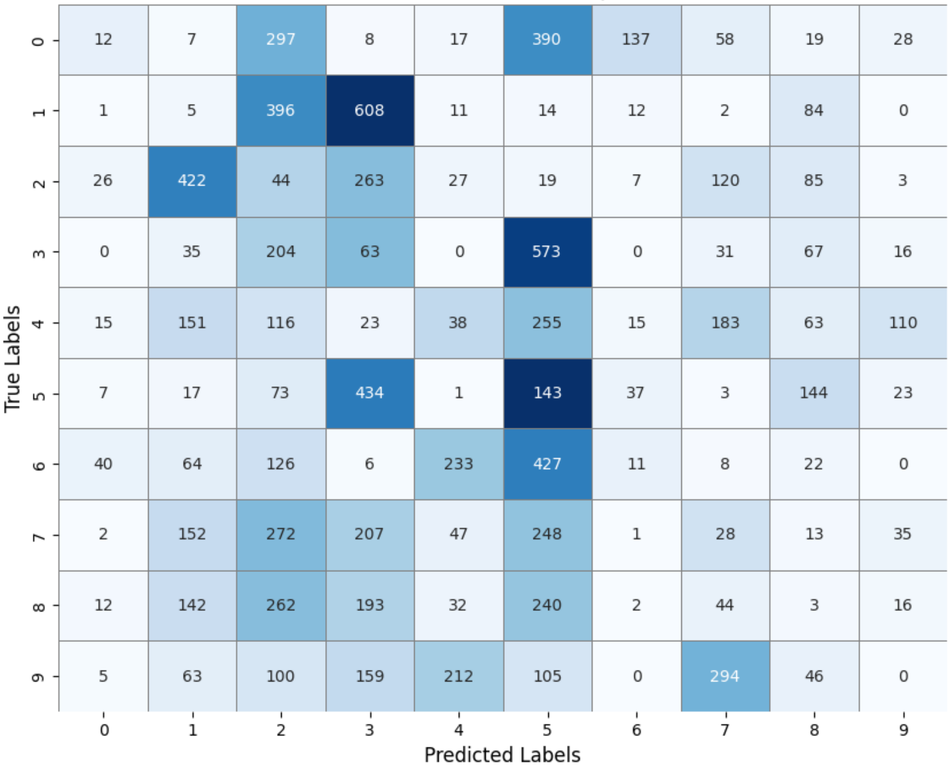
Confusion Matrix for eps=0.4



True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

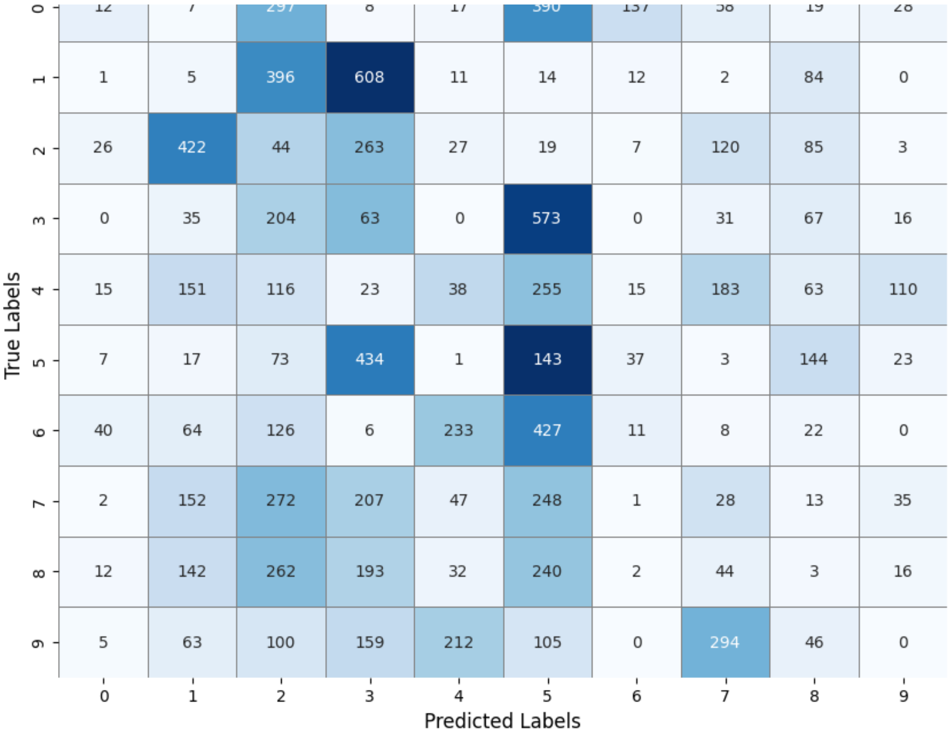
Confusion Matrix for eps=0.4



True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

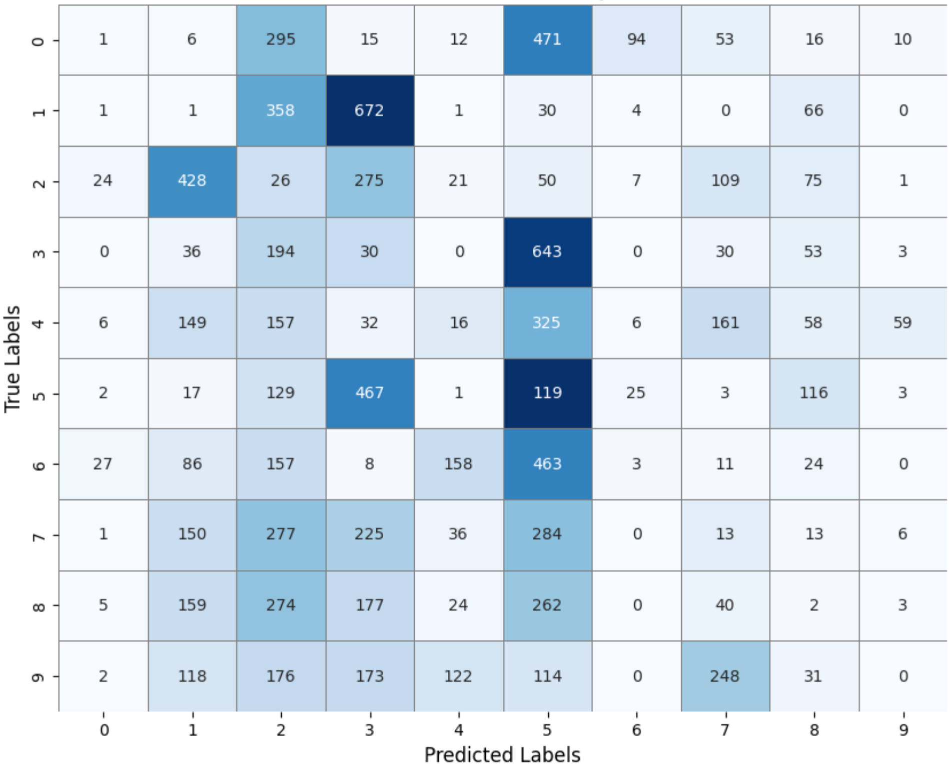
Confusion Matrix for eps=0.4



	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

Confusion Matrix for eps=0.5

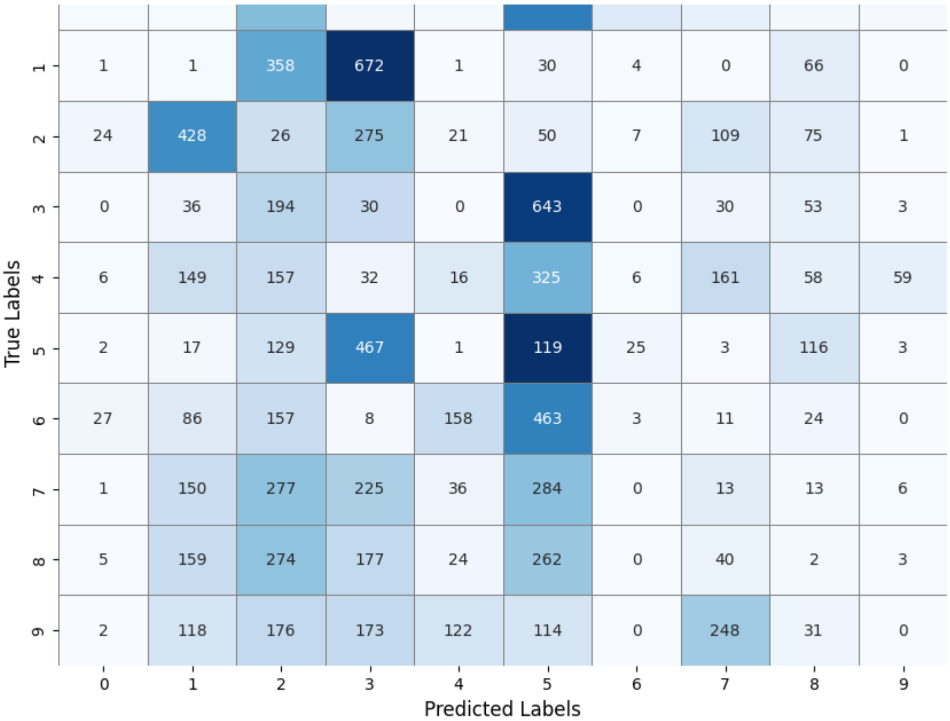


	0	1	2	3	4	5	6	7	8	9
0	1	6	295	15	12	471	94	53	16	10
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

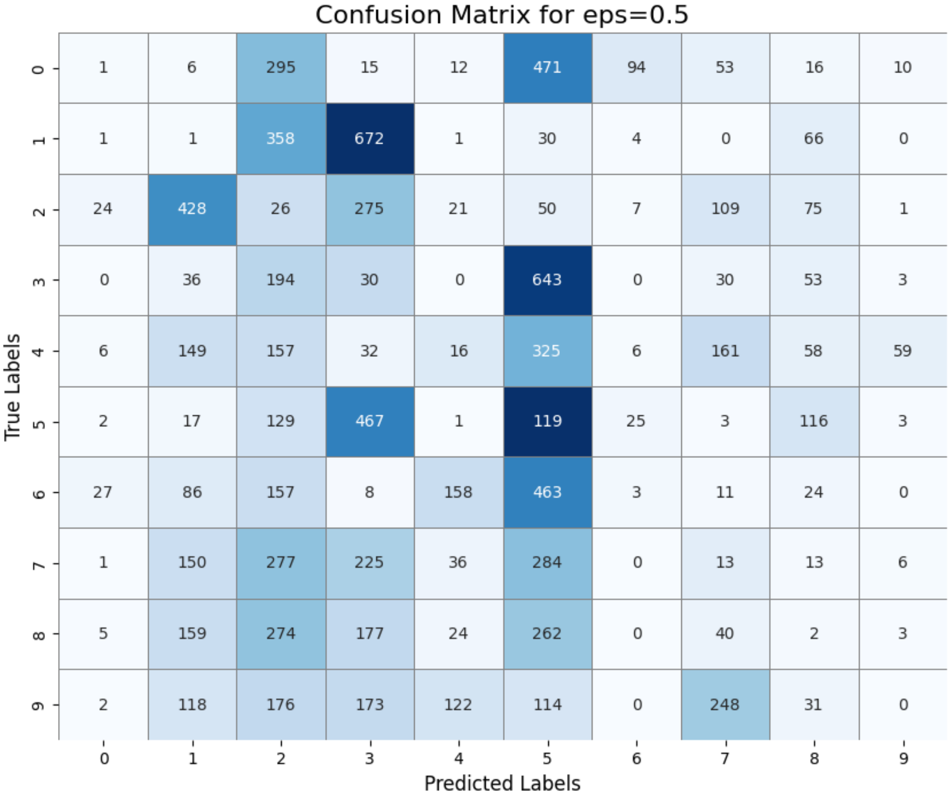
Confusion Matrix for eps=0.5

0	1	6	295	15	12	471	94	53	16	10



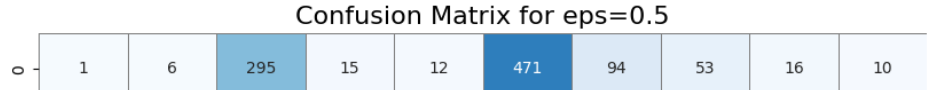
	0	1	2	3	4	5	6	7	8	9
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

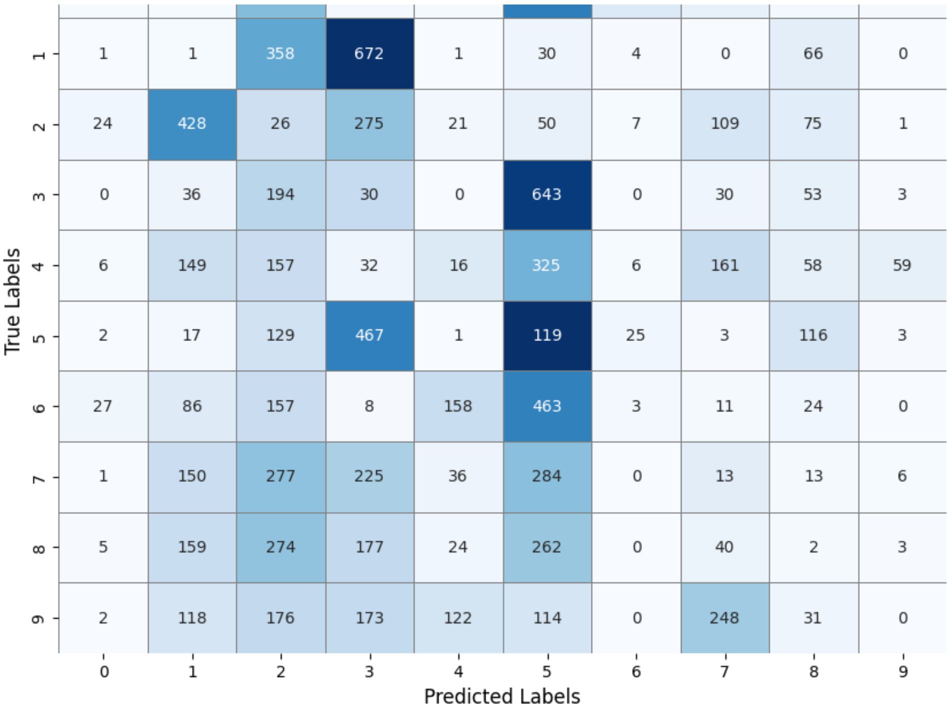


	0	1	2	3	4	5	6	7	8	9
0	1	6	295	15	12	471	94	53	16	10
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```



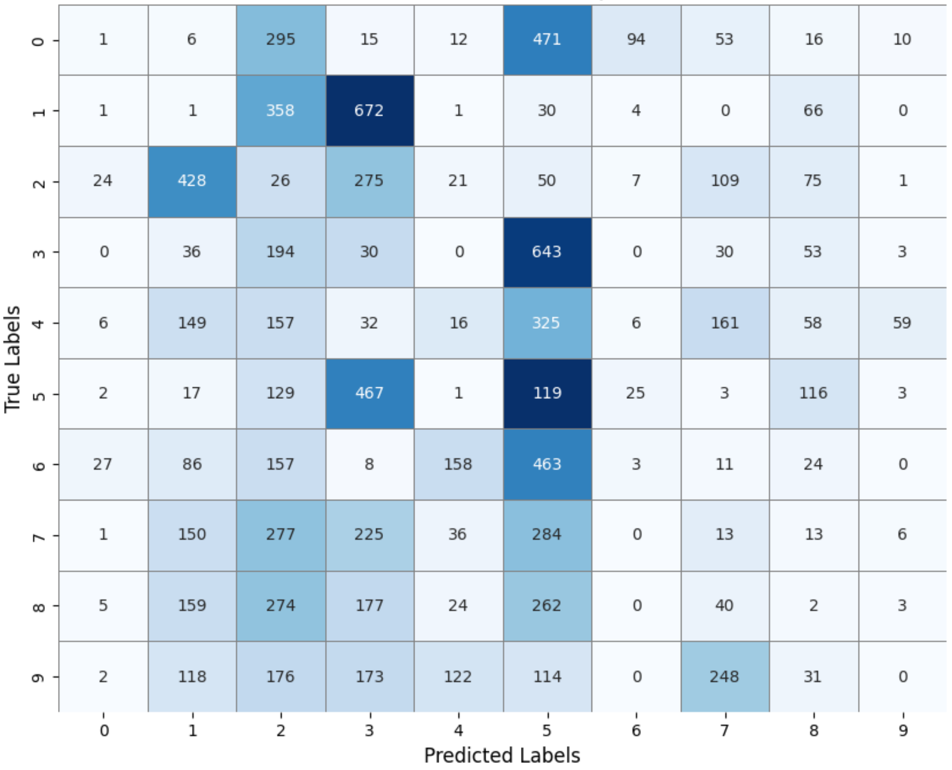
	0	1	2	3	4	5	6	7	8	9
0	1	6	295	15	12	471	94	53	16	10



	0	1	2	3	4	5	6	7	8	9
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

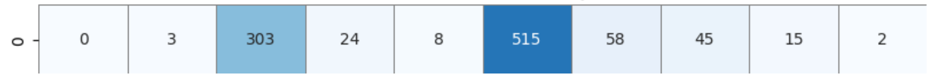
Confusion Matrix for eps=0.5



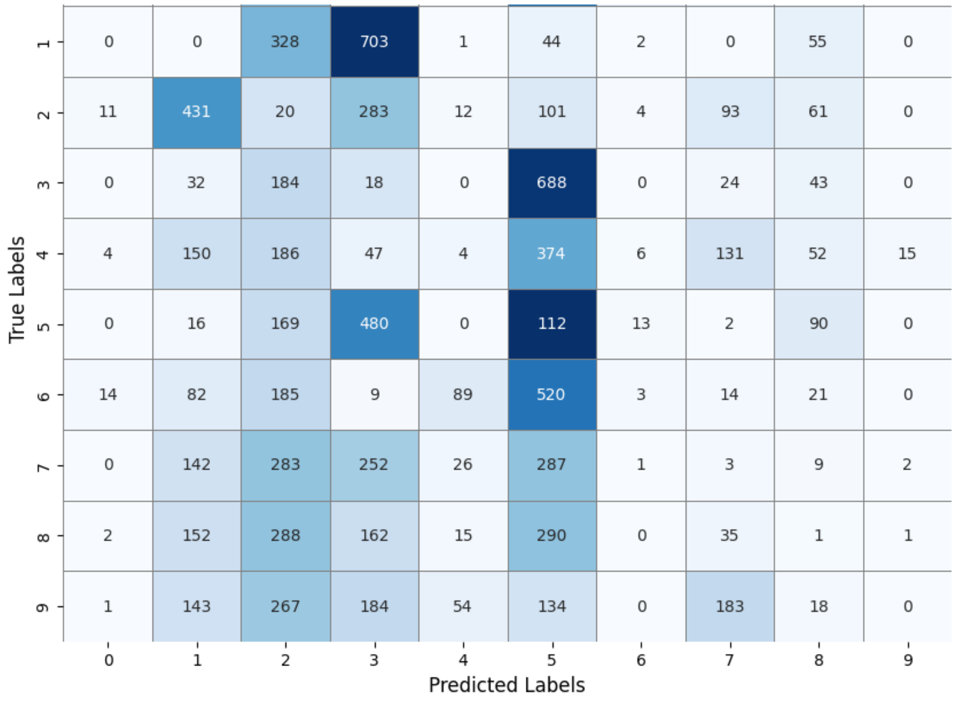
	0	1	2	3	4	5	6	7	8	9
0	1	6	295	15	12	471	94	53	16	10
1	1	1	358	672	1	30	4	0	66	0
2	24	428	26	275	21	50	7	109	75	1
3	0	36	194	30	0	643	0	30	53	3
4	6	149	157	32	16	325	6	161	58	59
5	2	17	129	467	1	119	25	3	116	3
6	27	86	157	8	158	463	3	11	24	0
7	1	150	277	225	36	284	0	13	13	6
8	5	159	274	177	24	262	0	40	2	3
9	2	118	176	173	122	114	0	248	31	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

Confusion Matrix for eps=0.6



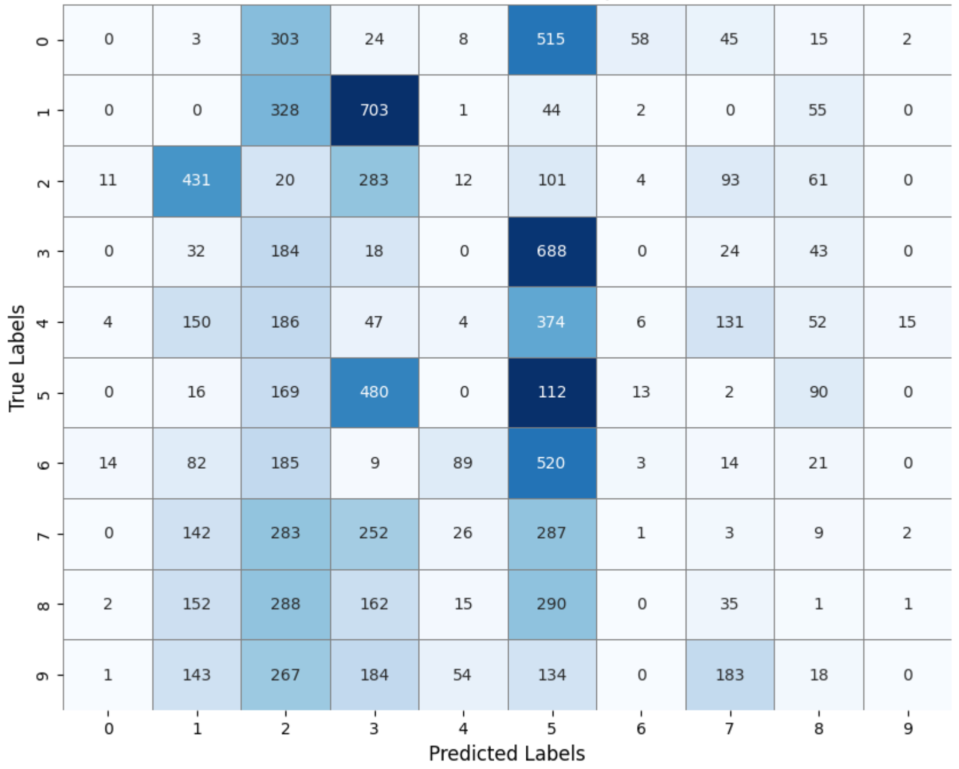
0	0	3	303	24	8	515	58	45	15	2



True Labels	0	1	2	3	4	5	6	7	8	9
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

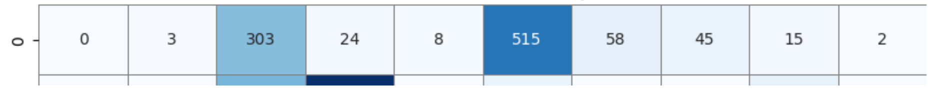
Confusion Matrix for eps=0.6



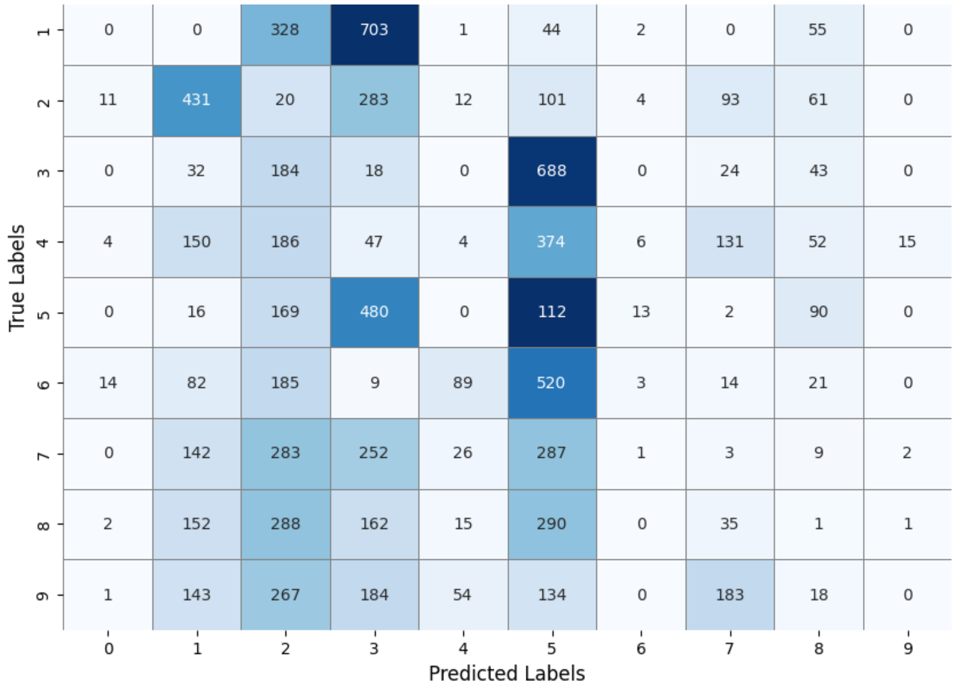
True Labels	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

Confusion Matrix for eps=0.6



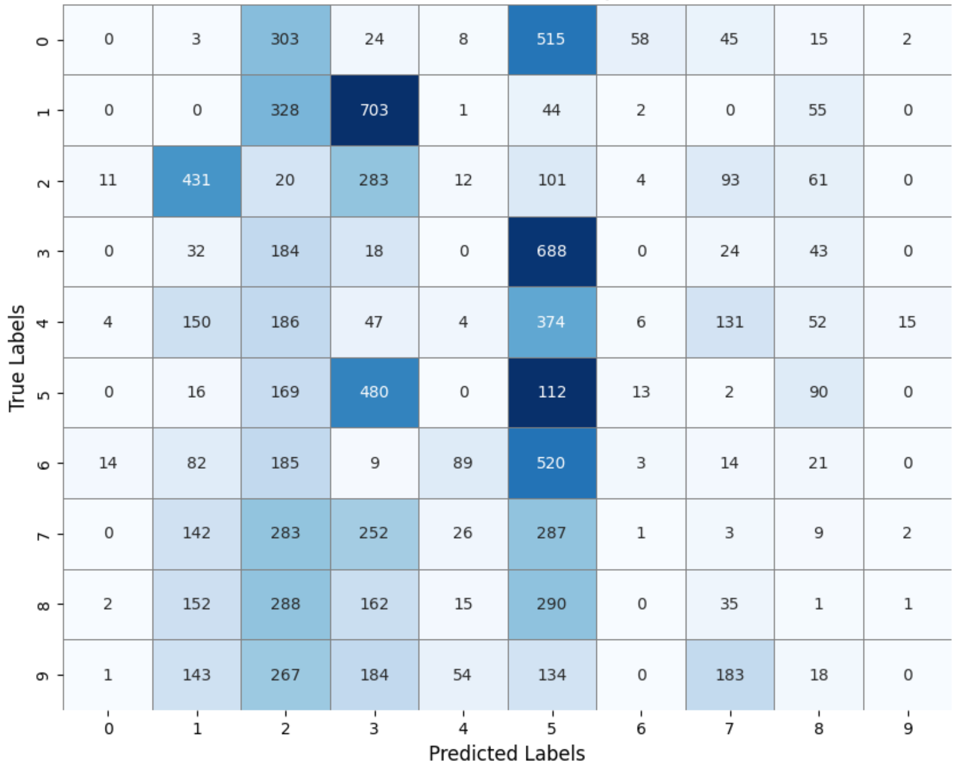
0	0	3	303	24	8	515	58	45	15	2



True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

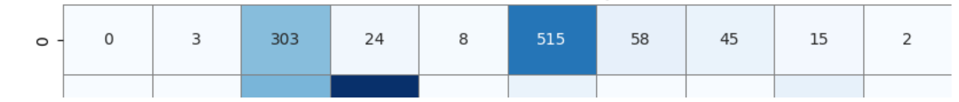
Confusion Matrix for eps=0.6



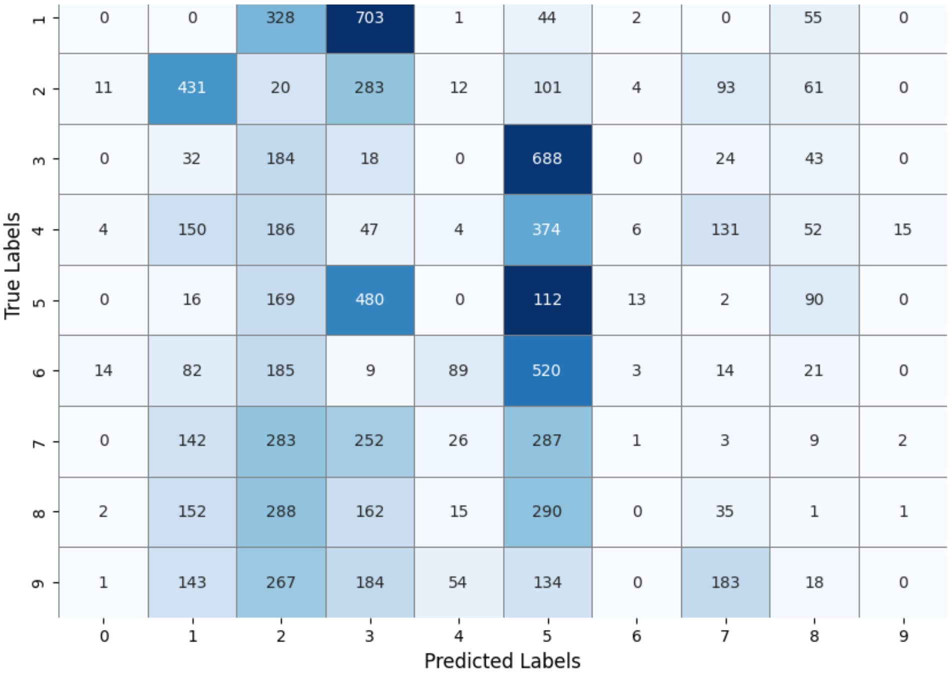
True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

Confusion Matrix for eps=0.6



True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2

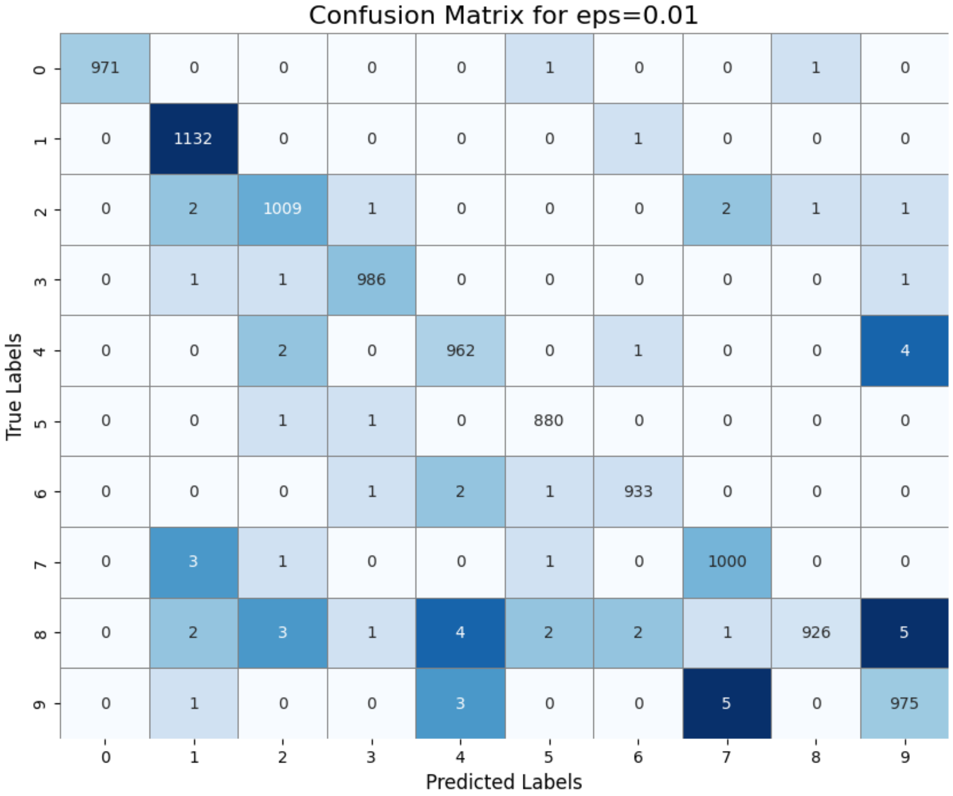


True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

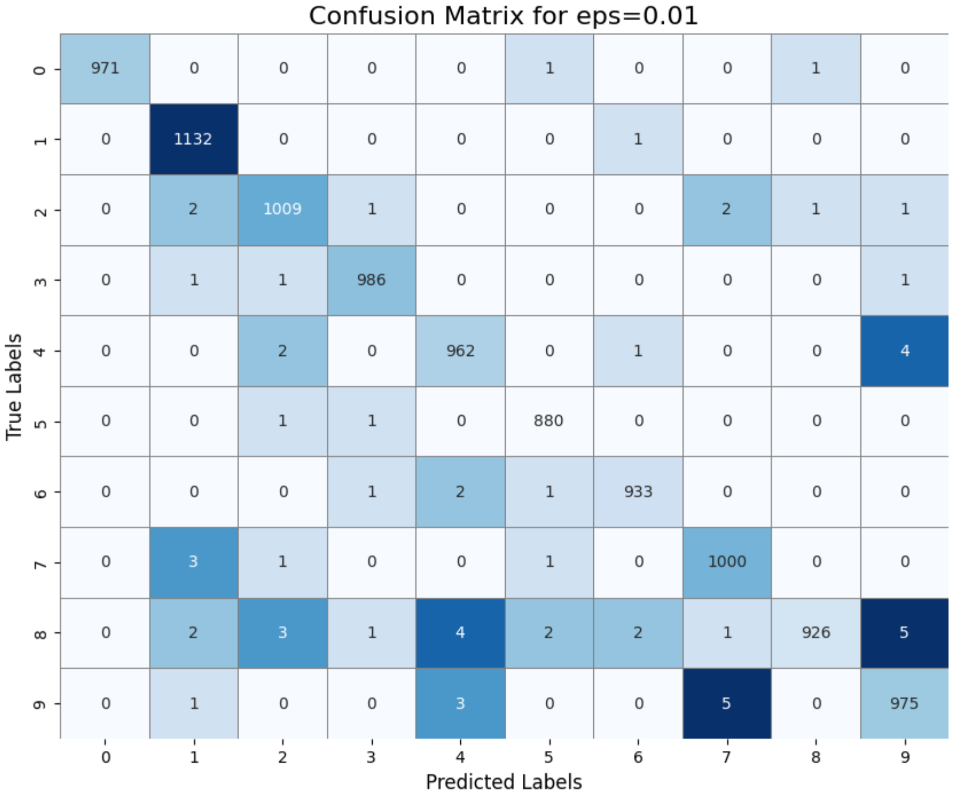
```
     eps  attack_num  total_correct  total_adv  \
0   0.01           1           9834       9834   
1   0.01           2           9834       9834   
2   0.01           3           9834       9834   
3   0.01           4           9834       9834   
4   0.01           5           9834       9834   
5   0.02           1           9834       9834   
6   0.02           2           9834       9834   
7   0.02           3           9834       9834   
8   0.02           4           9834       9834   
9   0.02           5           9834       9834   
10  0.03           1           9834       9834   
11  0.03           2           9834       9834   
12  0.03           3           9834       9834   
13  0.03           4           9834       9834   
14  0.03           5           9834       9834   
15  0.04           1           9834       9834   
16  0.04           2           9834       9834   
17  0.04           3           9834       9834   
18  0.04           4           9834       9834   
19  0.04           5           9834       9834   
20  0.05           1           9834       9834   
21  0.05           2           9834       9834   
22  0.05           3           9834       9834   
23  0.05           4           9834       9834   
24  0.05           5           9834       9834   
25  0.10           1           9834       9834   
26  0.10           2           9834       9834   
27  0.10           3           9834       9834   
28  0.10           4           9834       9834   
29  0.10           5           9834       9834   
30  0.20           1           9834       9834   
31  0.20           2           9834       9834   
32  0.20           3           9834       9834   
33  0.20           4           9834       9834   
34  0.20           5           9834       9834   
35  0.30           1           9834       9834   
36  0.30           2           9834       9834   
37  0.30           3           9834       9834   
38  0.30           4           9834       9834   
39  0.30           5           9834       9834   
40  0.40           1           9834       9834   
41  0.40           2           9834       9834   
42  0.40           3           9834       9834   
43  0.40           4           9834       9834   
44  0.40           5           9834       9834   
45  0.50           1           9834       9834   
46  0.50           2           9834       9834   
47  0.50           3           9834       9834   
48  0.50           4           9834       9834   
49  0.50           5           9834       9834   
50  0.60           1           9834       9834   
51  0.60           2           9834       9834   
52  0.60           3           9834       9834   
53  0.60           4           9834       9834   
```

```
54  0.60           5           9834       9834

                                     correct_adv_counts
0   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
1   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
2   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
3   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
4   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
5   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
6   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
7   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
8   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
9   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
10  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
11  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
12  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
13  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
14  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
15  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
16  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
17  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
18  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
19  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
20  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
21  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
22  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
23  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
24  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
25  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
26  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
27  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
28  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
29  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
30  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
31  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
32  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
33  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
34  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
35  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
36  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
37  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
38  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
39  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
40  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
41  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
42  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
43  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
44  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
45  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
46  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
47  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
48  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
49  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
50  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
51  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
52  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
53  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
54  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
<ipython-input-56-9fea085cbc61>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
        # Apply the attack to generate adversarial examples
        x_adv = attack.generate(x=correct_examples,y=correct_labels)

        # Predict the labels of the adversarial examples
        y_adv = np.argmax(classifier.predict(x_adv), axis=1)

        nb_correct_adv_pred = np.sum(y_adv == correct_labels)

        print(f"Adversarial test data: eps:{eps}")
        print("Correctly classified: {}".format(nb_correct_adv_pred))
        print("Incorrectly classified: {}".format(len(correct_examples)-nb_correct_adv_pred))

        # For clean examples
        y_pred_clean = np.argmax(classifier.predict(correct_examples), axis=1)
        accuracy_clean = np.sum(y_pred_clean == correct_labels) / len(correct_labels)
        print(f"accuracy_clean:{accuracy_clean}")
        # Inside the loop for each eps
        # You have already calculated this for adversarial examples:
        # nb_correct_adv_pred = np.sum(y_adv == correct_labels)
        # Now calculate the accuracy for adversarial examples
        accuracy_adv = nb_correct_adv_pred / len(correct_labels)

        print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")
        print(f"Adversarial test data: eps:{eps}")
        print(f"Accuracy on adversarial examples: {accuracy_adv * 100:.2f}%")


        # Calculate the confusion matrix
        cm = confusion_matrix(correct_labels, y_adv, labels=range(10))
        # Draw and save the confusion matrix
        fig, ax = plt.subplots(figsize=(10, 8))

        # Create a mask for the diagonal elements
        mask = np.eye(len(cm), dtype=bool)

        # Plot the heatmap for off-diagonal elements using the mask
        # Use a professional color palette like 'Blues'
        sns.heatmap(cm, mask=mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecol
        # Plot the heatmap for diagonal elements using the inverse of the mask
        # Use the same color palette for consistency
        sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, lineco
        # Labels, title and ticks
        label_names = [f'{i}' for i in range(10)]
        ax.set_xlabel('Predicted Labels', fontsize=12)
        ax.set_ylabel('True Labels', fontsize=12)
        ax.set_title(f'Confusion Matrix for eps={eps}', fontsize=16)
        ax.set_xticklabels(label_names)
        ax.set_yticklabels(label_names)
        image_filename = f'target_confusion_matrix_eps_{eps}_attack_{attack_num}.png'
        plt.savefig(image_filename, bbox_inches='tight')
        plt.show()  # Close the figure to avoid displaying it in the notebook

        # Save the results in the DataFrame
        results_df = results_df.append({
            'eps': eps,
            'attack_num': attack_num,
            'total_correct': len(correct_labels),
            'total_adv': len(y_adv),
            'correct_adv_counts': dict(zip(*np.unique(y_adv, return_counts=True)))
        }, ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('/content/drive/MyDrive/ColabNotebooks/5attack_withtruelabel_adv_results.csv', index=False)

# Print the DataFrame
print(results_df)
```

```
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```

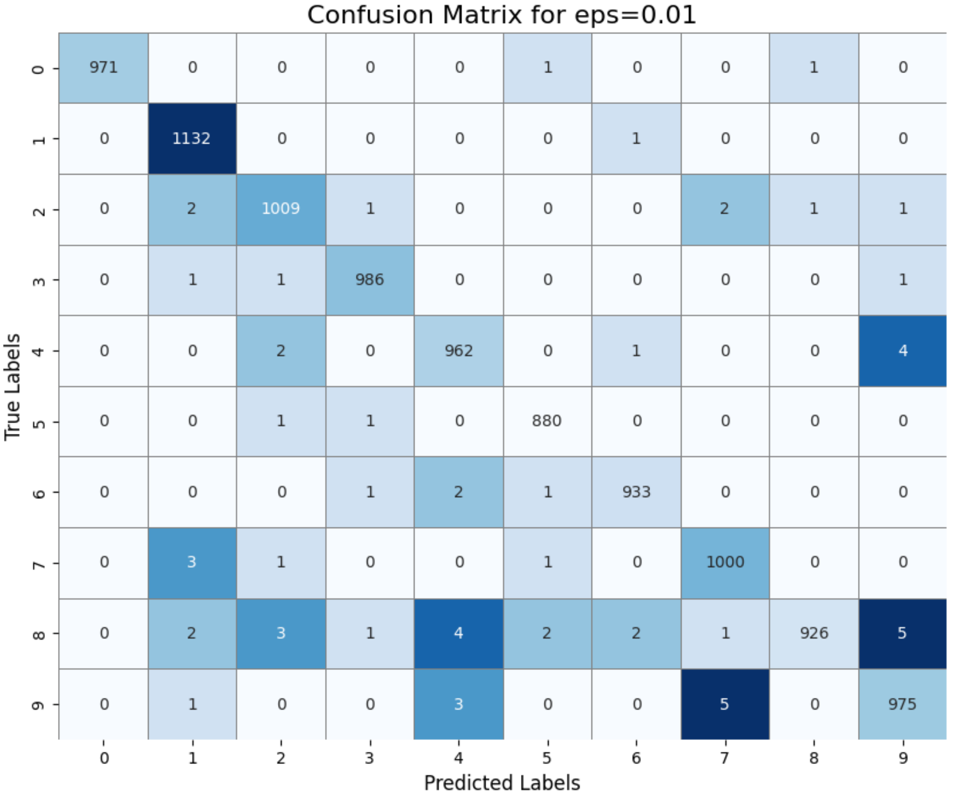


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



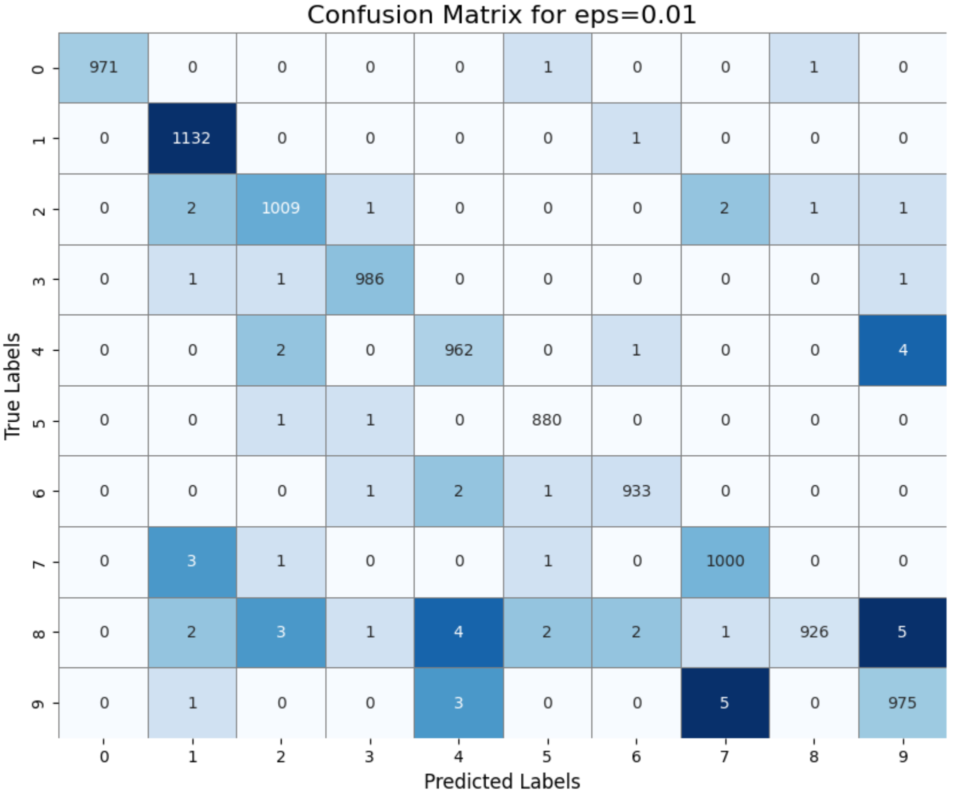
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
```

```
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```



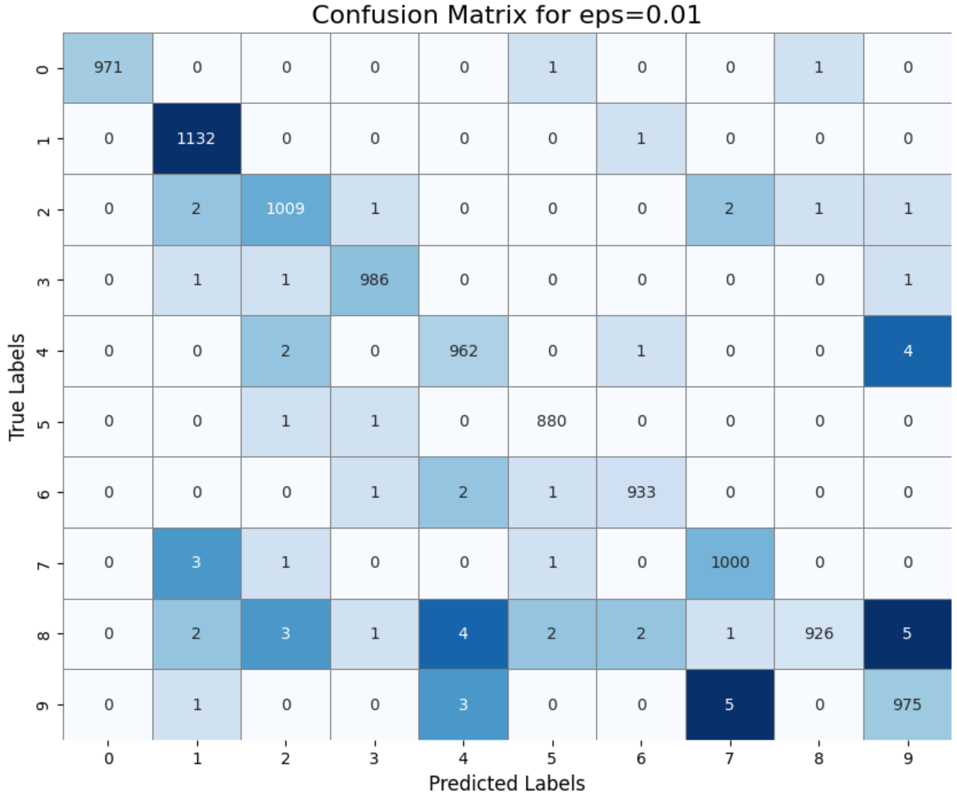
	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```

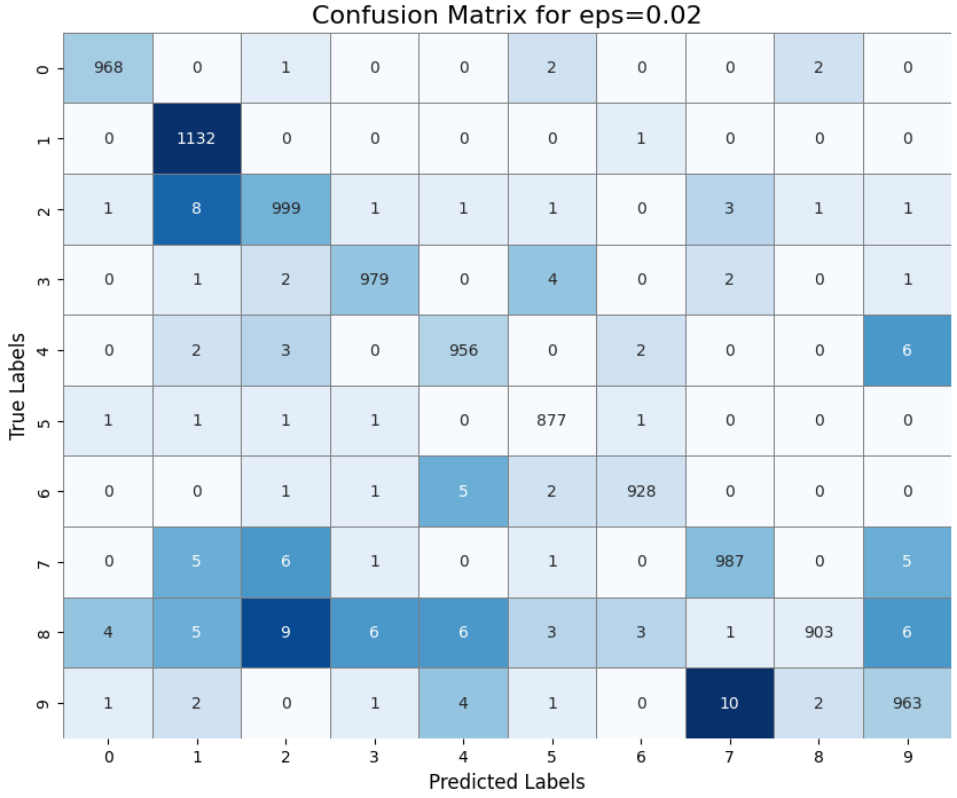


	0	1	2	3	4	5	6	7	8	9
0	971	0	0	0	0	1	0	0	1	0
1	0	1132	0	0	0	0	1	0	0	0
2	0	2	1009	1	0	0	0	2	1	1
3	0	1	1	986	0	0	0	0	0	1
4	0	0	2	0	962	0	1	0	0	4
5	0	0	1	1	0	880	0	0	0	0
6	0	0	0	1	2	1	933	0	0	0
7	0	3	1	0	0	1	0	1000	0	0
8	0	2	3	1	4	2	2	1	926	5
9	0	1	0	0	3	0	0	5	0	975

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.01
Adversarial test data: eps:0.01
Correctly classified: 9774
Incorrectly classified: 60
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```

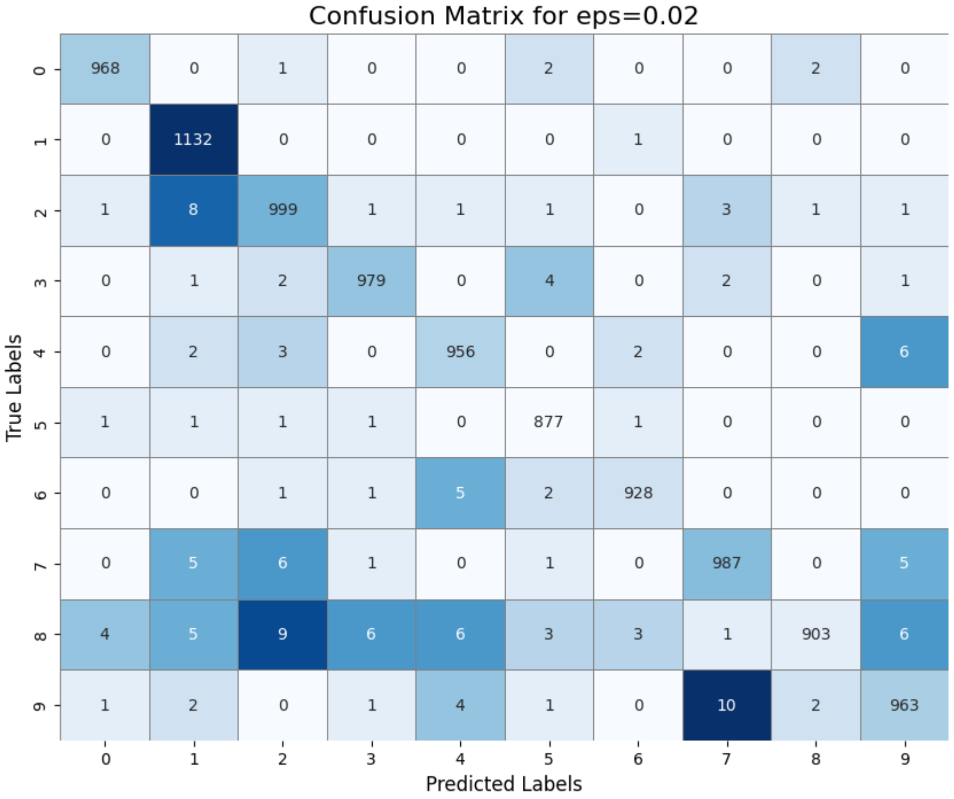


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



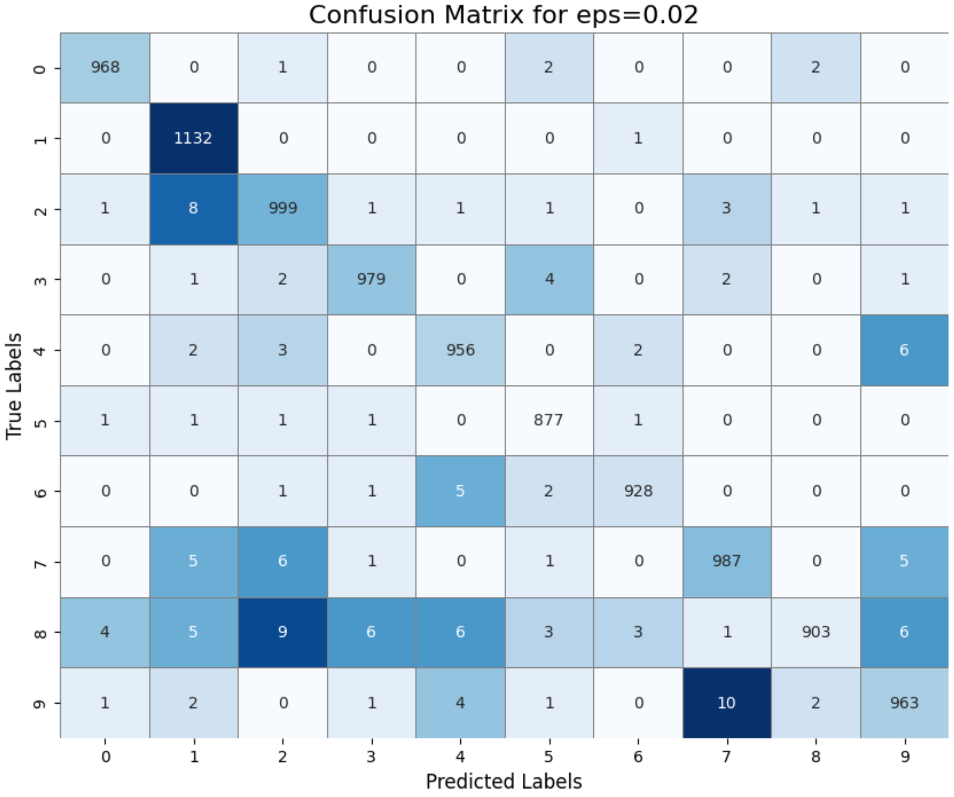
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
```

```
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



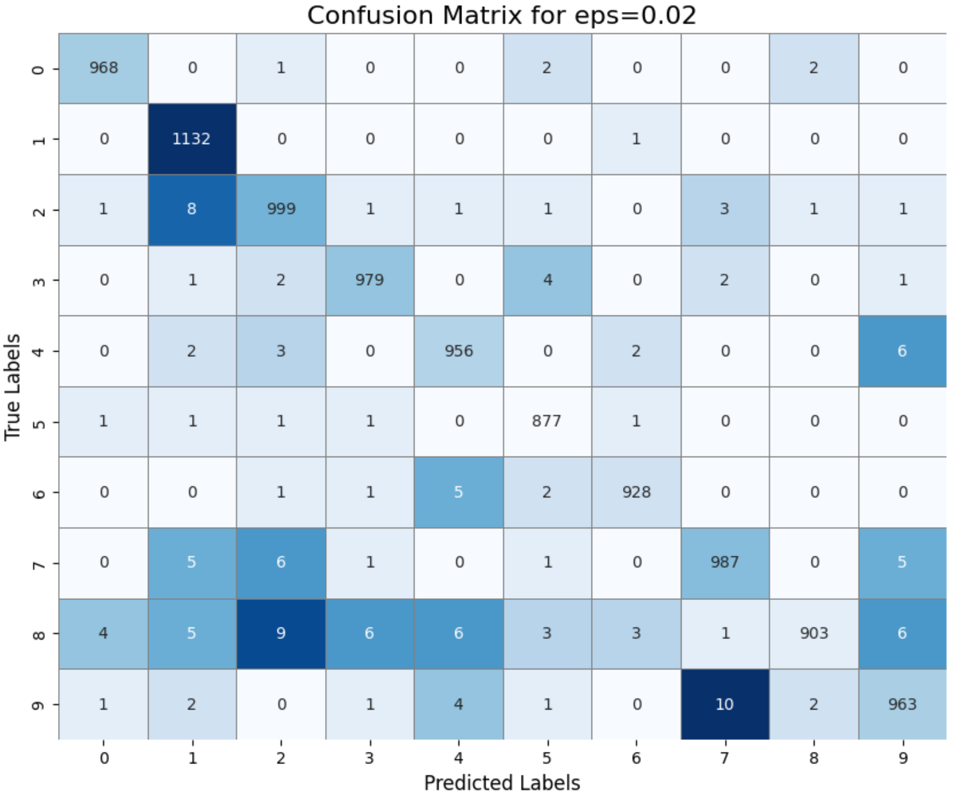
	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

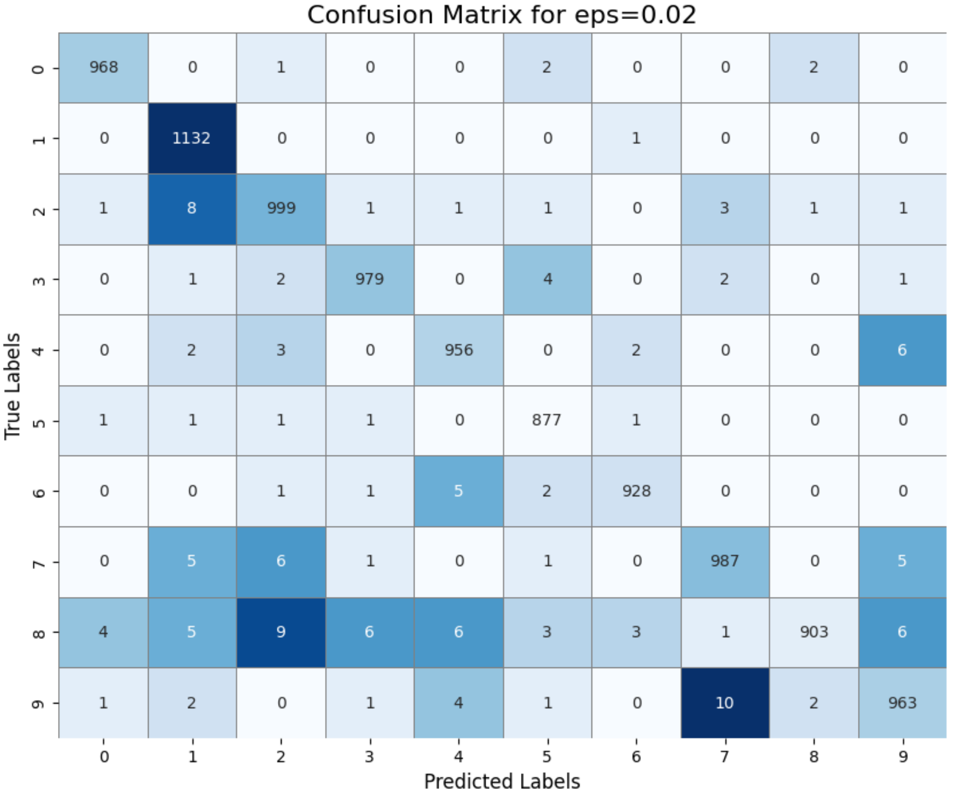
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



Confusion Matrix for eps=0.02

	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.02
Correctly classified: 9692
Incorrectly classified: 142
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```

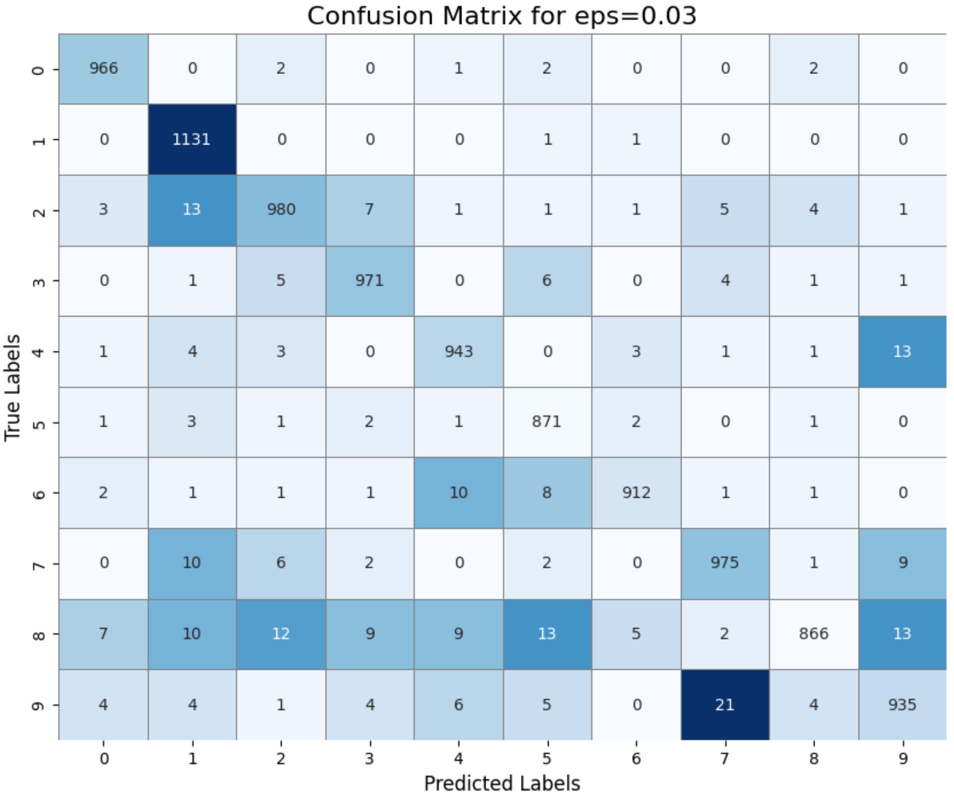


Confusion Matrix for eps=0.02

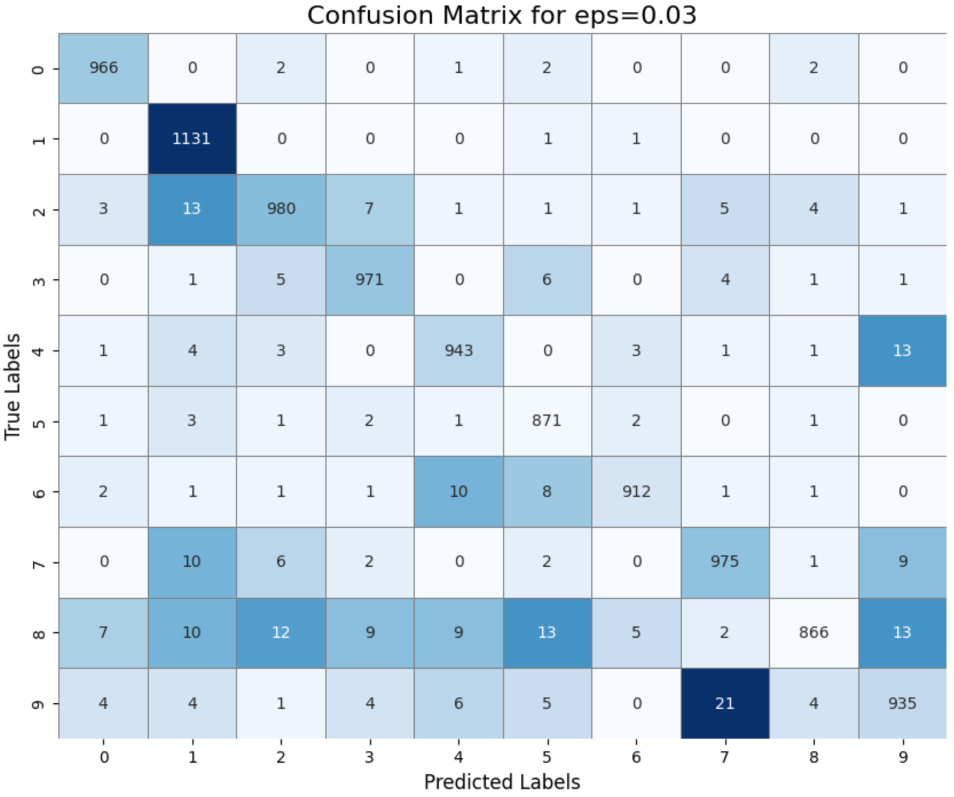
	0	1	2	3	4	5	6	7	8	9
0	968	0	1	0	0	2	0	0	2	0
1	0	1132	0	0	0	0	1	0	0	0
2	1	8	999	1	1	1	0	3	1	1
3	0	1	2	979	0	4	0	2	0	1
4	0	2	3	0	956	0	2	0	0	6
5	1	1	1	1	0	877	1	0	0	0
6	0	0	1	1	5	2	928	0	0	0
7	0	5	6	1	0	1	0	987	0	5
8	4	5	9	6	6	3	3	1	903	6
9	1	2	0	1	4	1	0	10	2	963

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
```

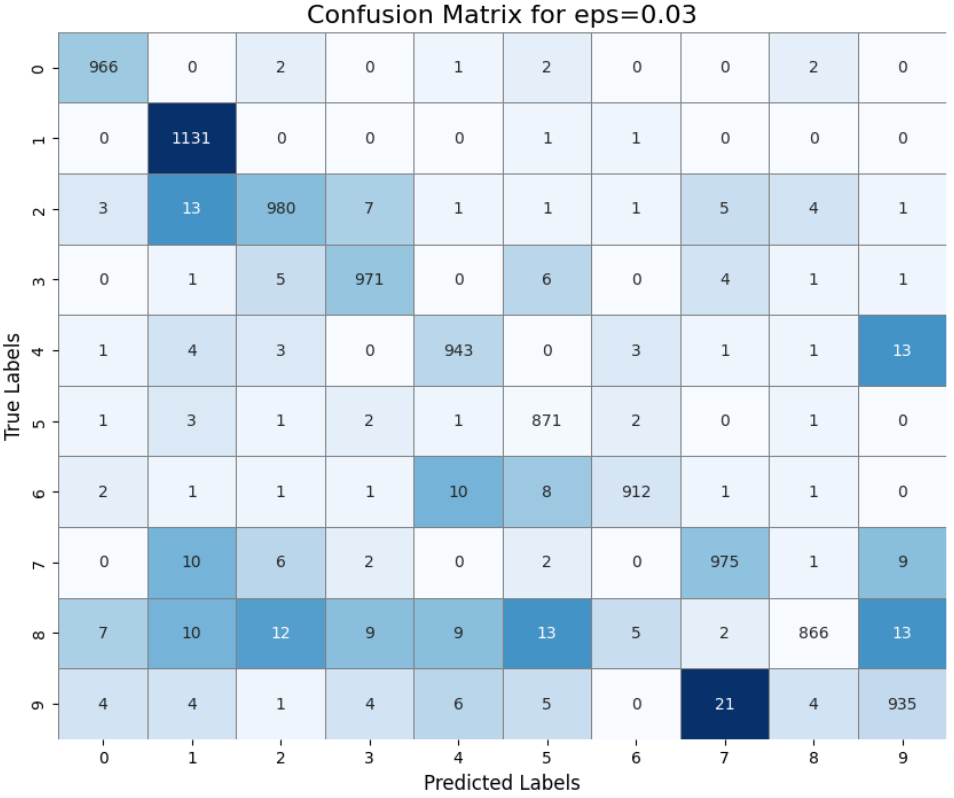
```
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```

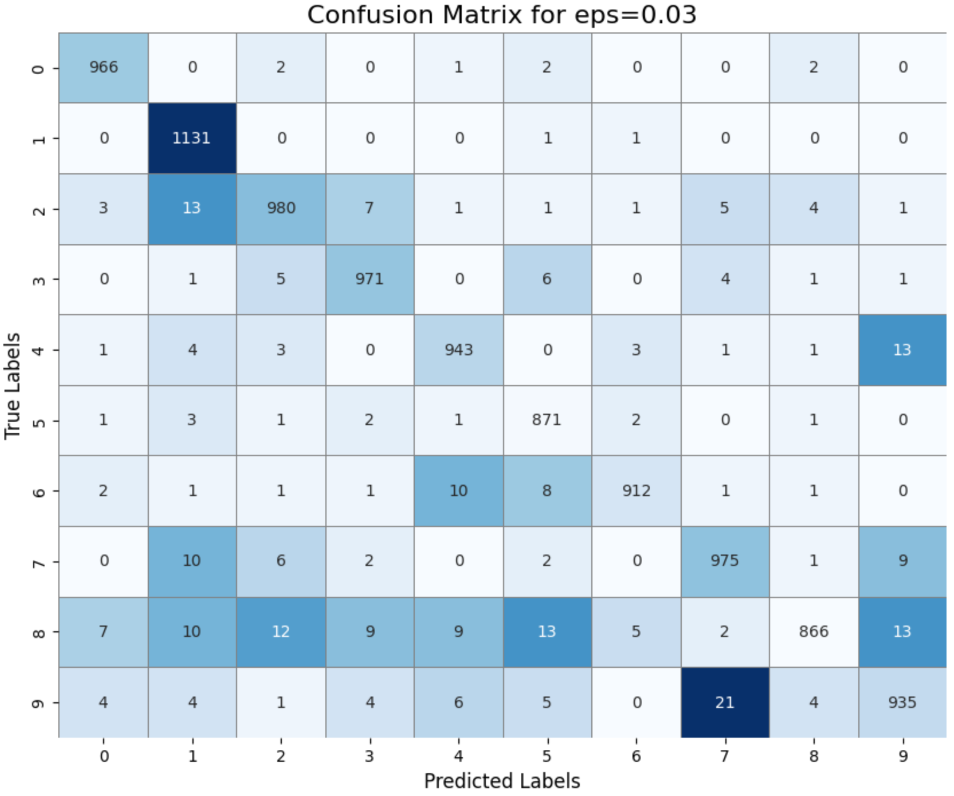


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

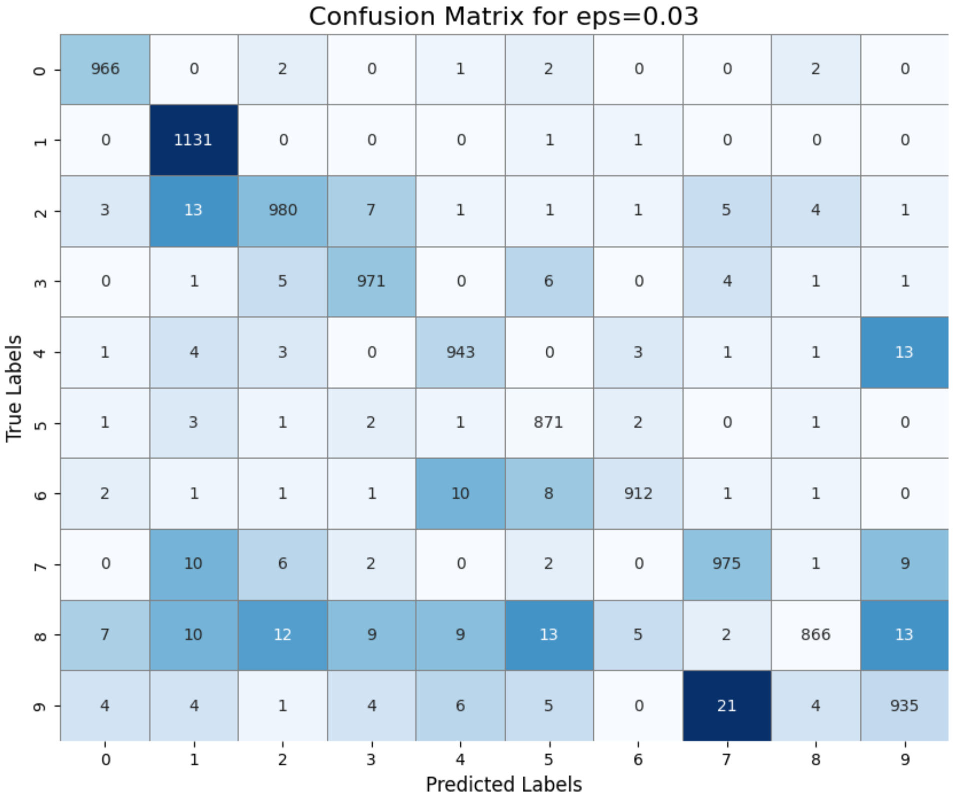
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```



	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

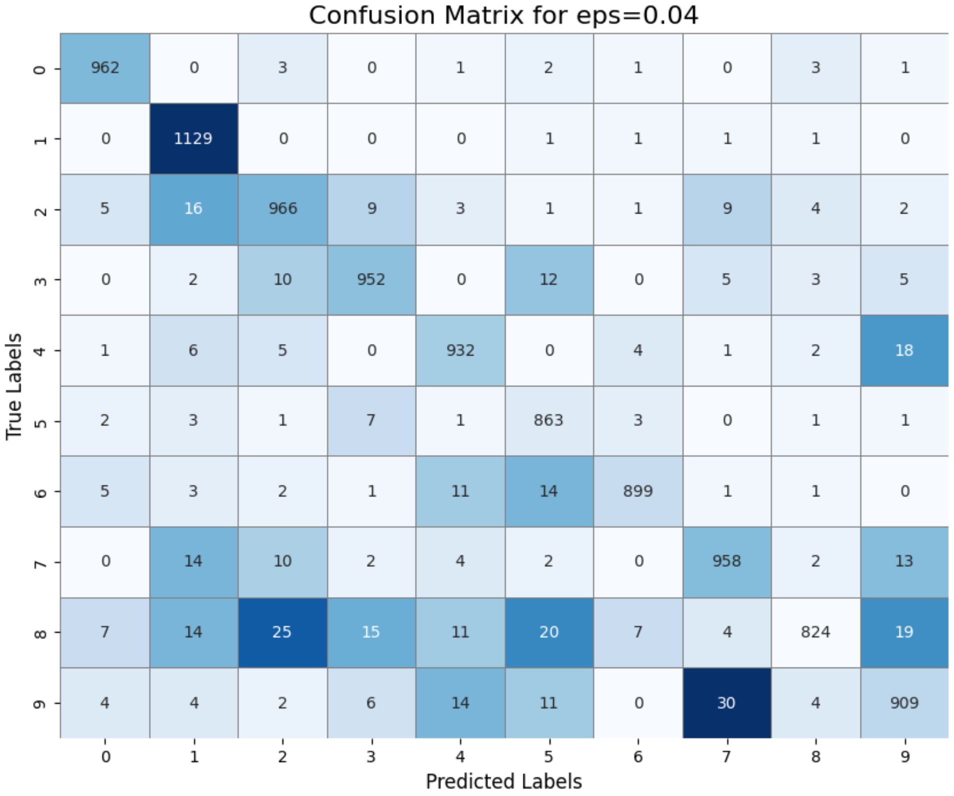
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.03
Correctly classified: 9550
Incorrectly classified: 284
```

```
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```


Confusion Matrix for eps=0.03

	0	1	2	3	4	5	6	7	8	9
0	966	0	2	0	1	2	0	0	2	0
1	0	1131	0	0	0	1	1	0	0	0
2	3	13	980	7	1	1	1	5	4	1
3	0	1	5	971	0	6	0	4	1	1
4	1	4	3	0	943	0	3	1	1	13
5	1	3	1	2	1	871	2	0	1	0
6	2	1	1	1	10	8	912	1	1	0
7	0	10	6	2	0	2	0	975	1	9
8	7	10	12	9	9	13	5	2	866	13
9	4	4	1	4	6	5	0	21	4	935

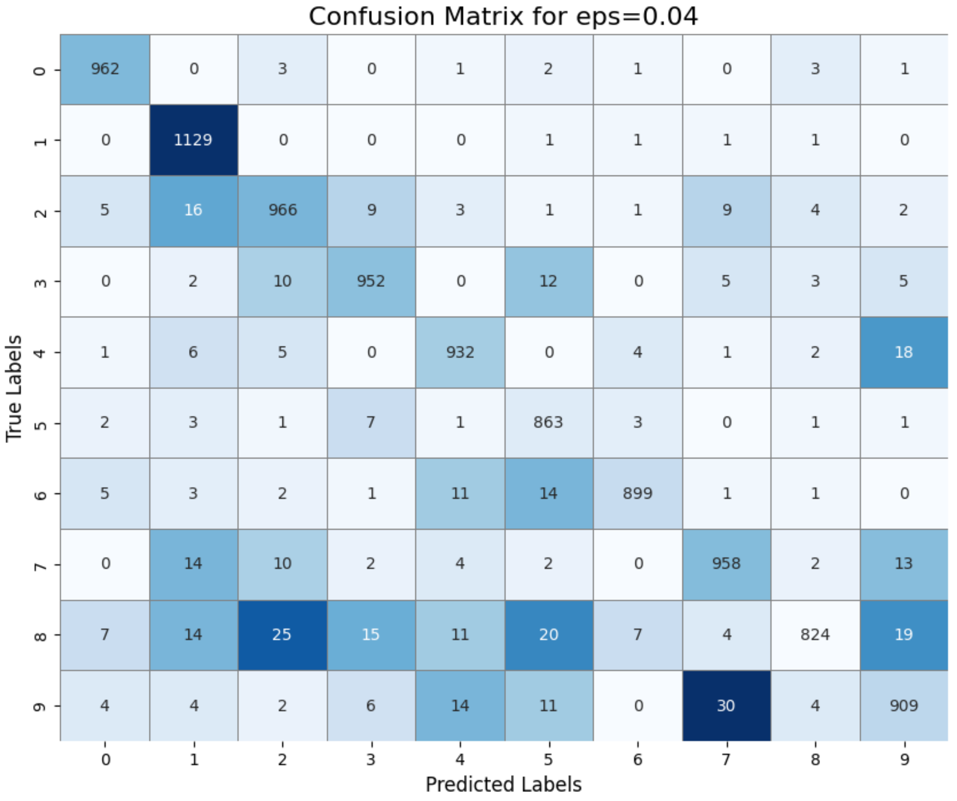
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```


Confusion Matrix for eps=0.04

	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

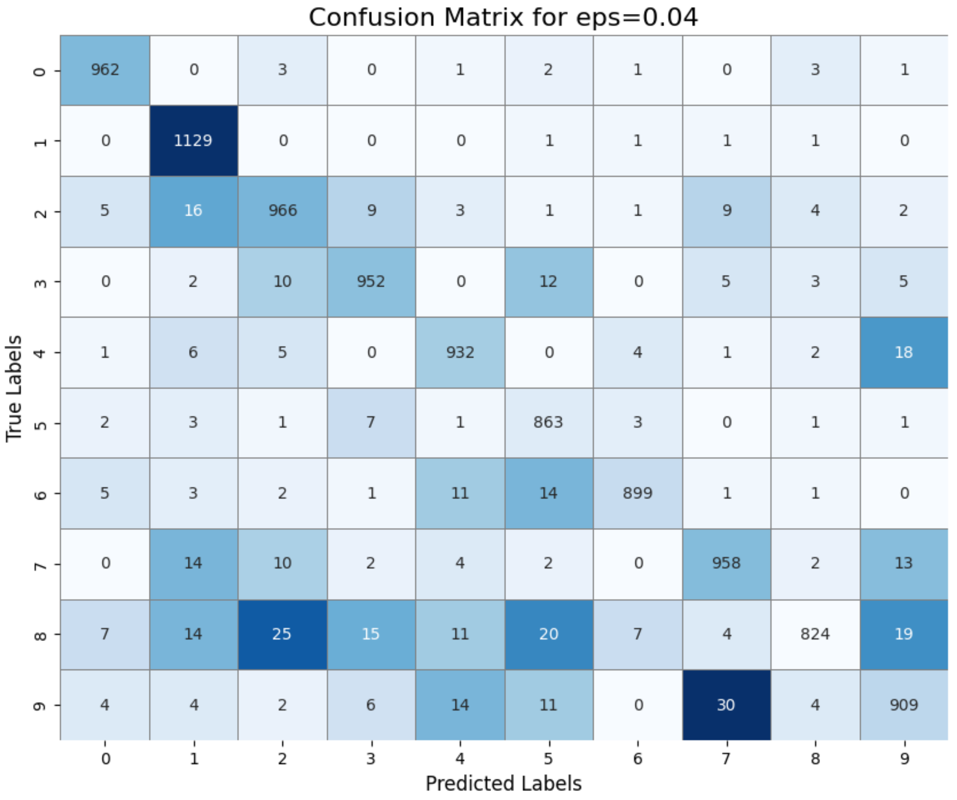
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
```

```
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```


Confusion Matrix for eps=0.04

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

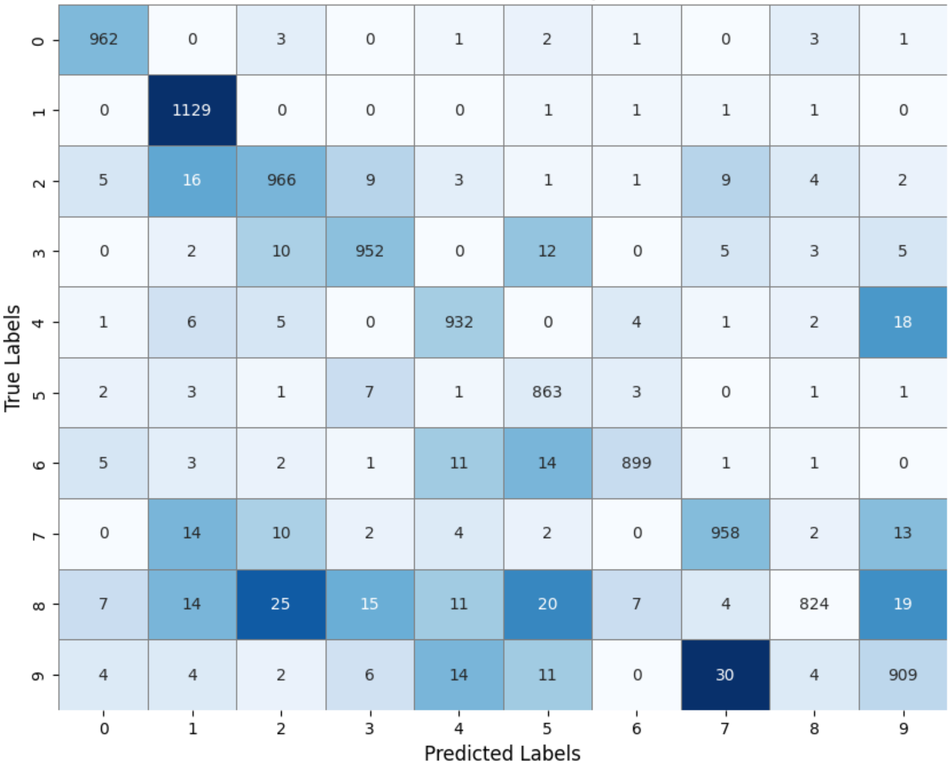

Confusion Matrix for eps=0.04

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
```

```
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

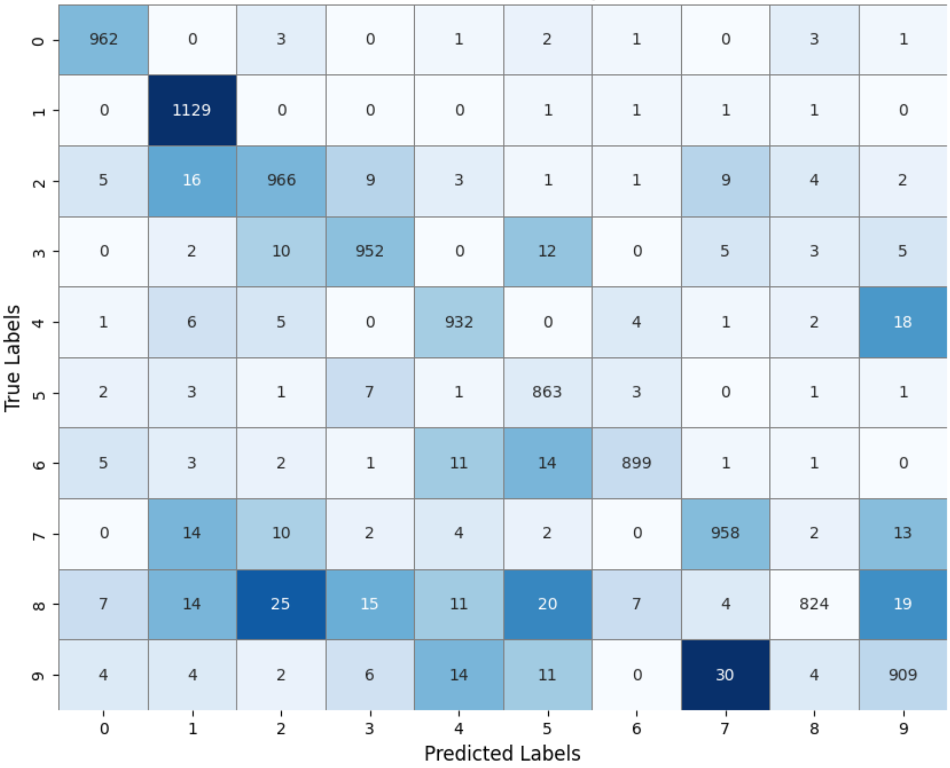
Confusion Matrix for eps=0.04



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.04
Correctly classified: 9394
Incorrectly classified: 440
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%
```

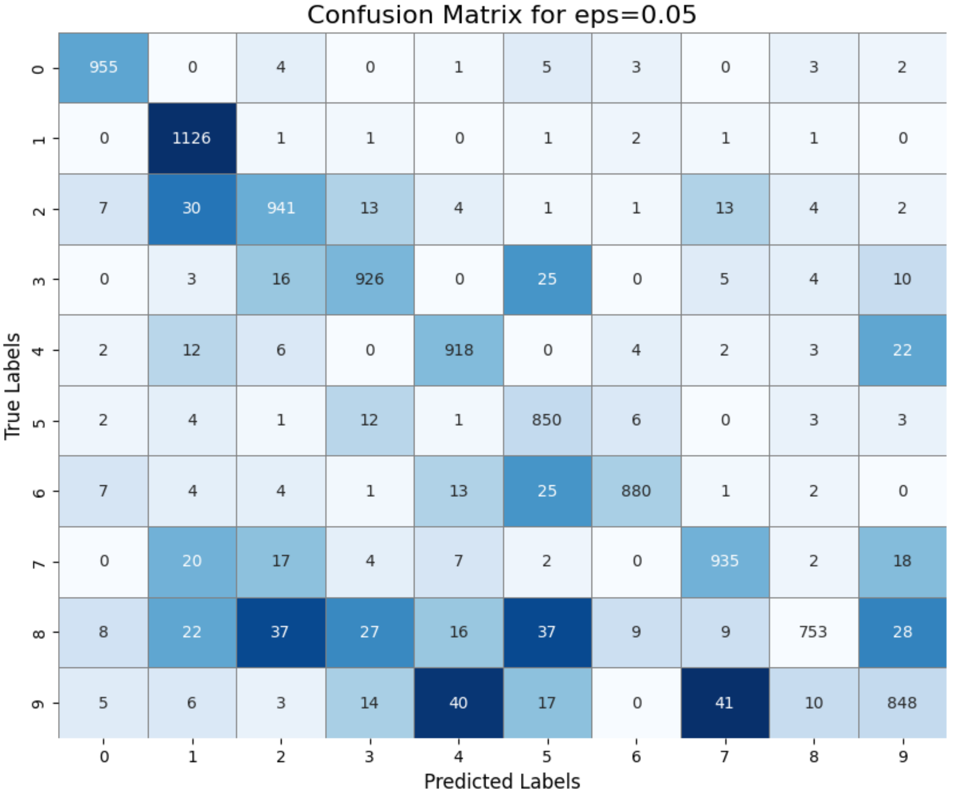
Confusion Matrix for eps=0.04



	0	1	2	3	4	5	6	7	8	9
0	962	0	3	0	1	2	1	0	3	1
1	0	1129	0	0	0	1	1	1	1	0
2	5	16	966	9	3	1	1	9	4	2
3	0	2	10	952	0	12	0	5	3	5
4	1	6	5	0	932	0	4	1	2	18
5	2	3	1	7	1	863	3	0	1	1
6	5	3	2	1	11	14	899	1	1	0
7	0	14	10	2	4	2	0	958	2	13
8	7	14	25	15	11	20	7	4	824	19
9	4	4	2	6	14	11	0	30	4	909

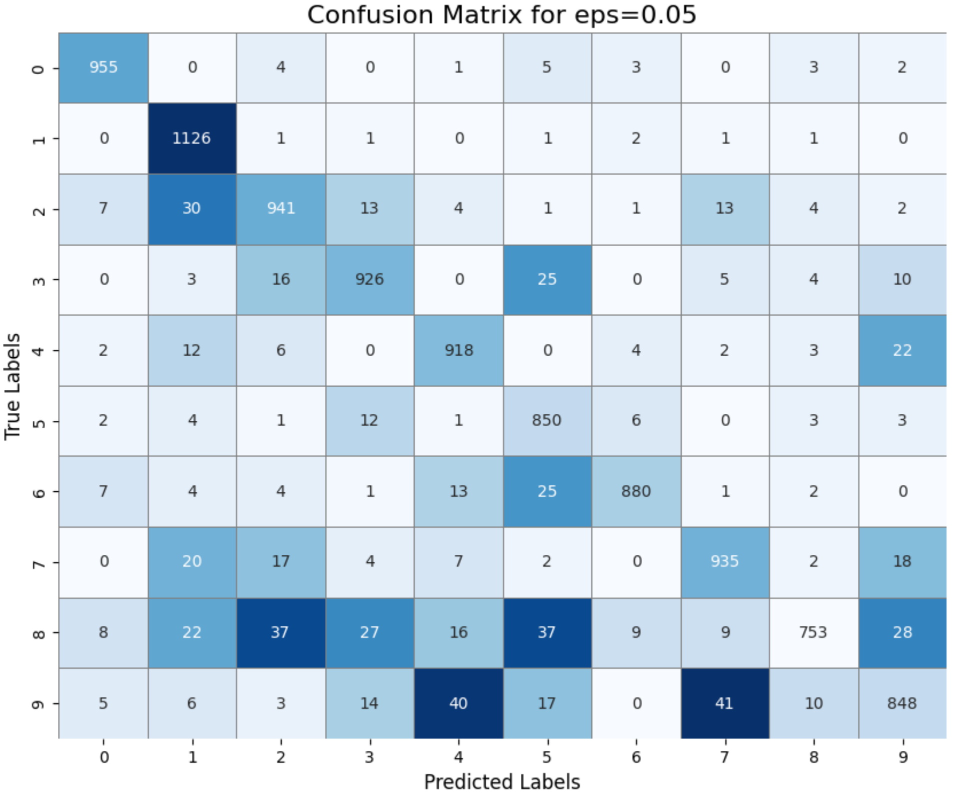
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
```

```
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



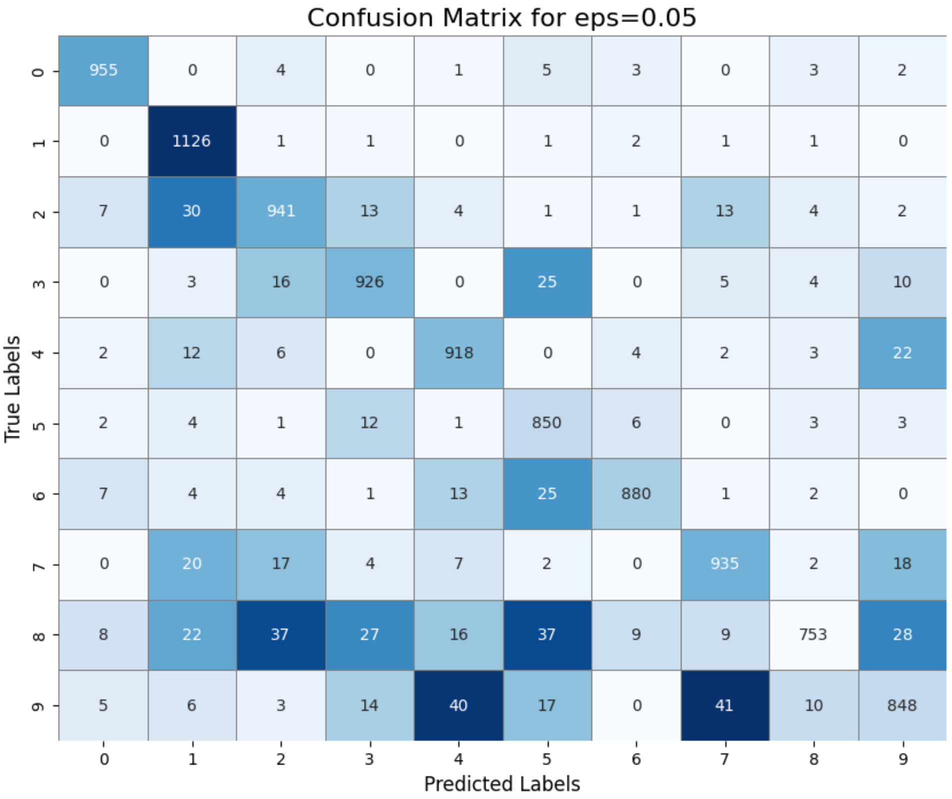
	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



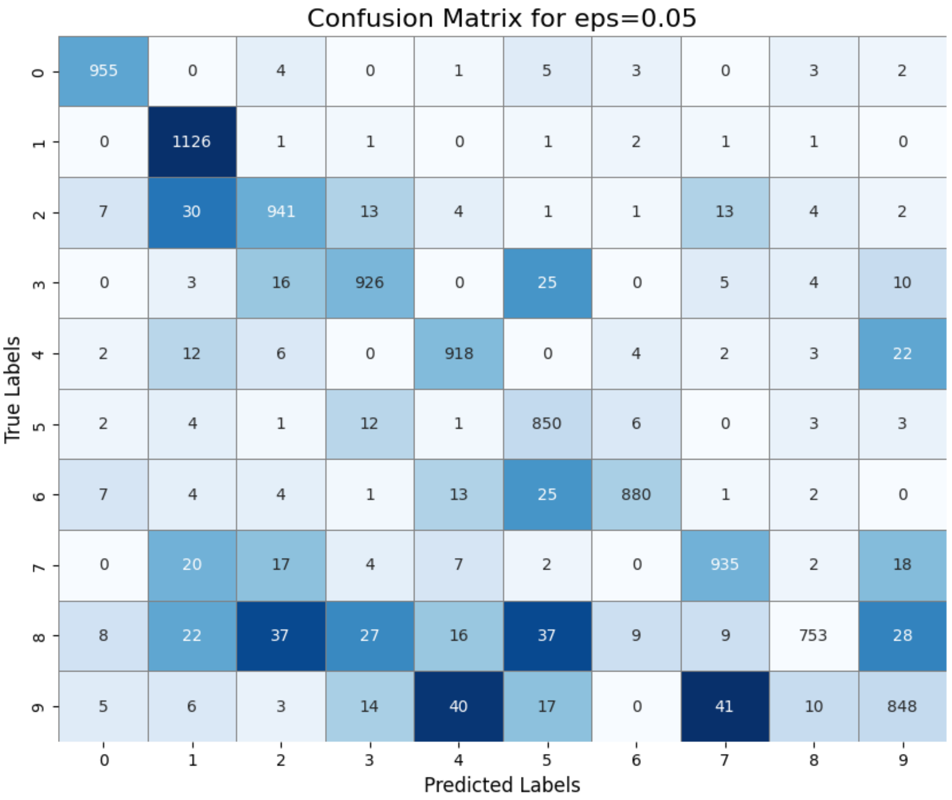
	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

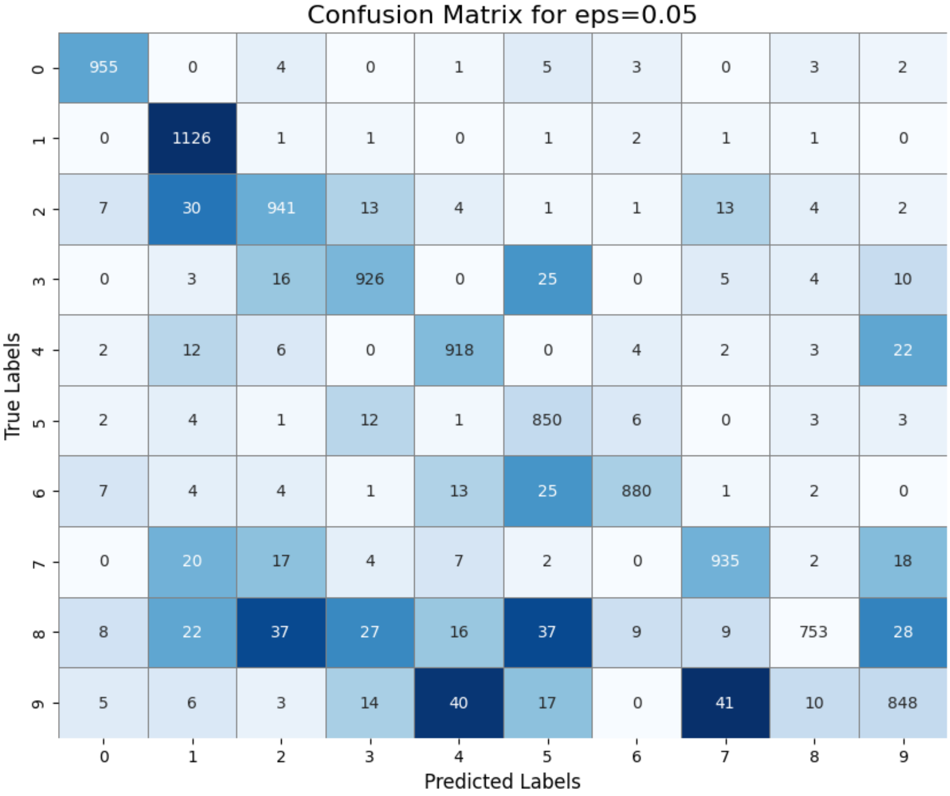
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.05
Correctly classified: 9132
Incorrectly classified: 702
accuracy_clean:1.0
Accuracy on clean data: 100.00%
```

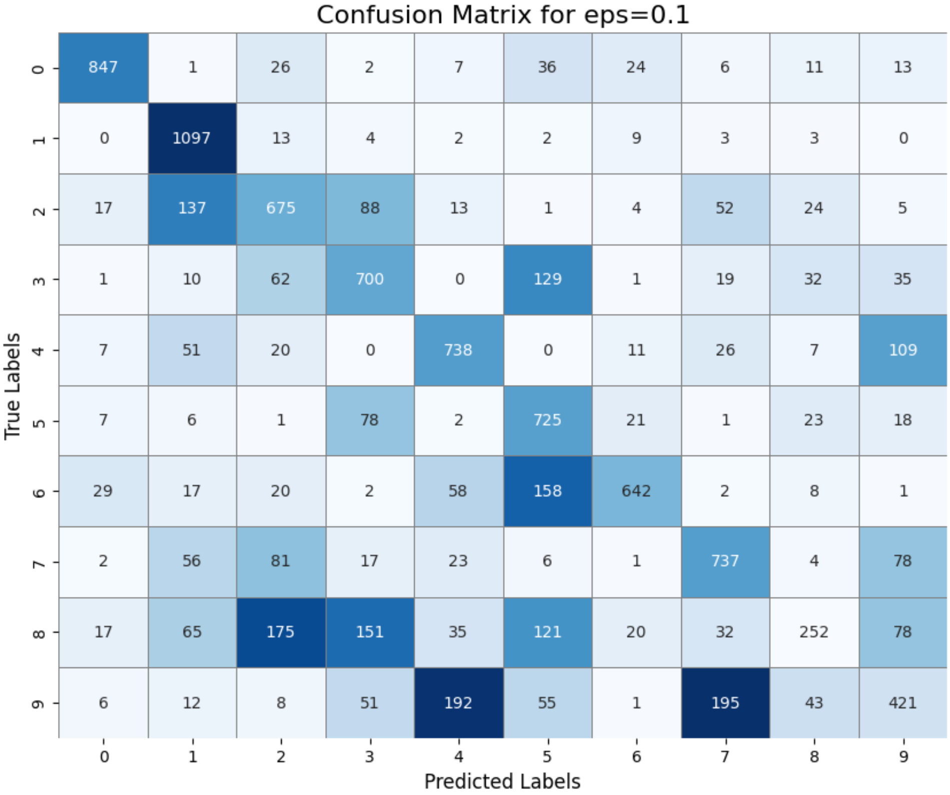
```
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.86%
```



Confusion Matrix for eps=0.05

	0	1	2	3	4	5	6	7	8	9
0	955	0	4	0	1	5	3	0	3	2
1	0	1126	1	1	0	1	2	1	1	0
2	7	30	941	13	4	1	1	13	4	2
3	0	3	16	926	0	25	0	5	4	10
4	2	12	6	0	918	0	4	2	3	22
5	2	4	1	12	1	850	6	0	3	3
6	7	4	4	1	13	25	880	1	2	0
7	0	20	17	4	7	2	0	935	2	18
8	8	22	37	27	16	37	9	9	753	28
9	5	6	3	14	40	17	0	41	10	848

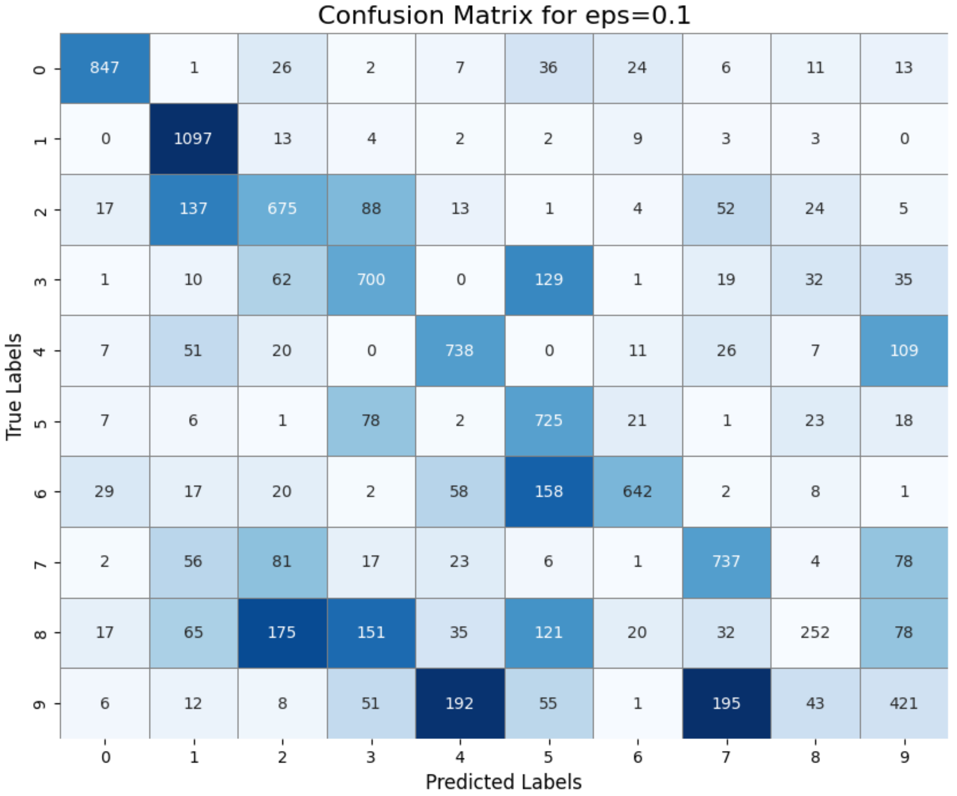
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```



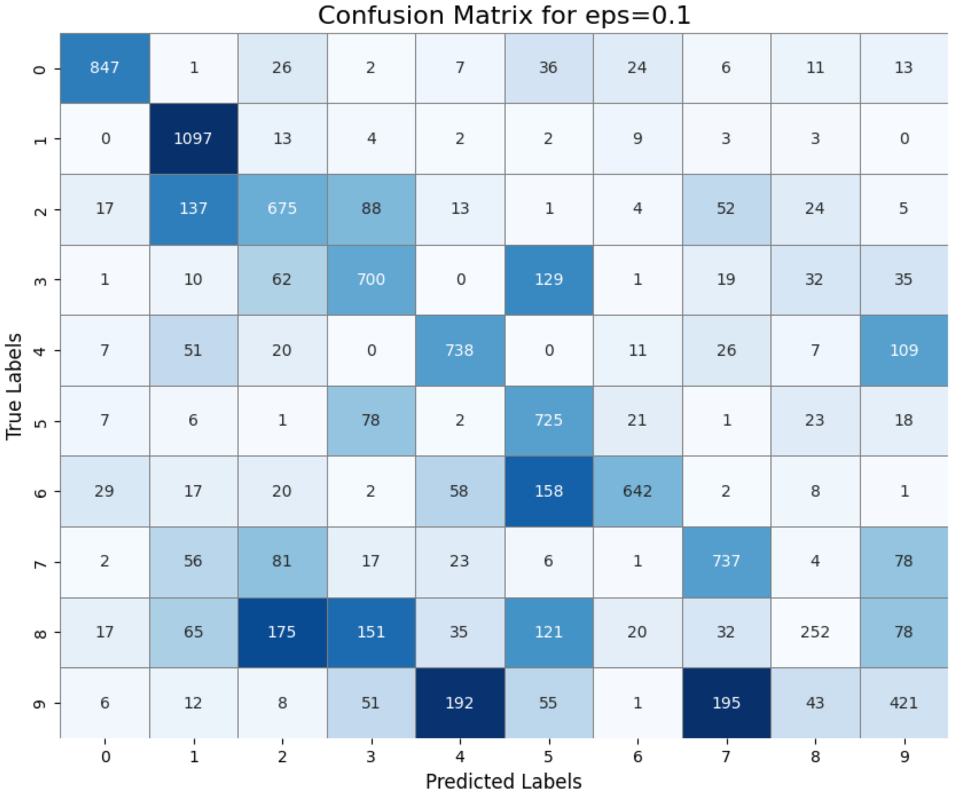
Confusion Matrix for eps=0.1

	0	1	2	3	4	5	6	7	8	9
0	847	1	26	2	7	36	24	6	11	13
1	0	1097	13	4	2	2	9	3	3	0
2	17	137	675	88	13	1	4	52	24	5
3	1	10	62	700	0	129	1	19	32	35
4	7	51	20	0	738	0	11	26	7	109
5	7	6	1	78	2	725	21	1	23	18
6	29	17	20	2	58	158	642	2	8	1
7	2	56	81	17	23	6	1	737	4	78
8	17	65	175	151	35	121	20	32	252	78
9	6	12	8	51	192	55	1	195	43	421

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

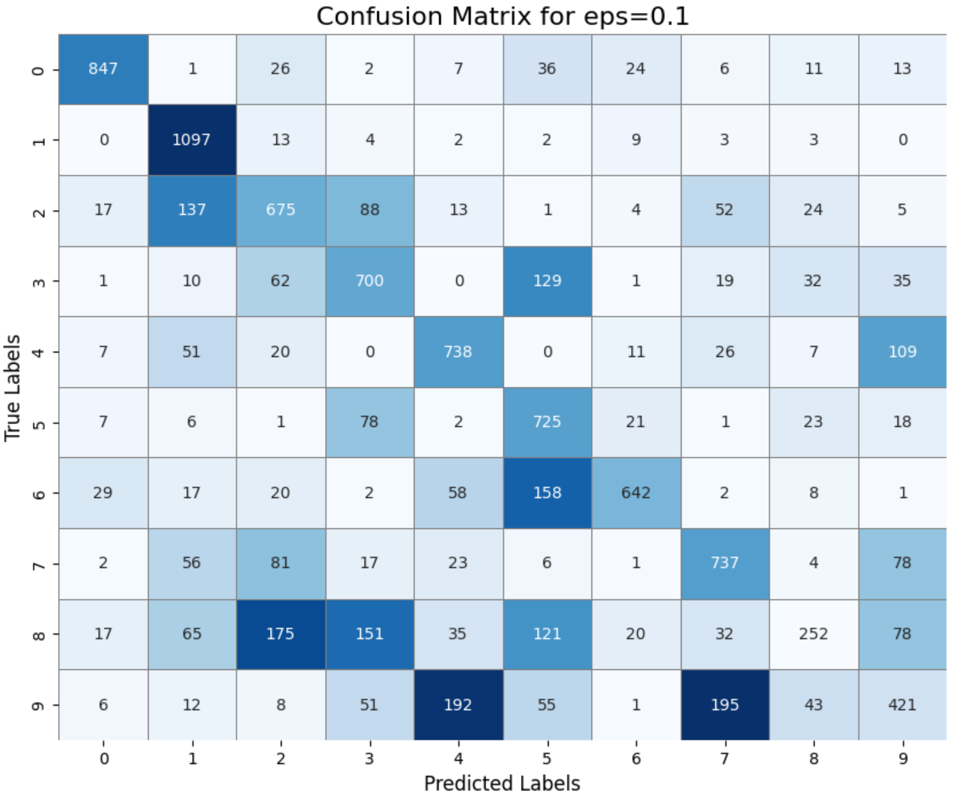


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

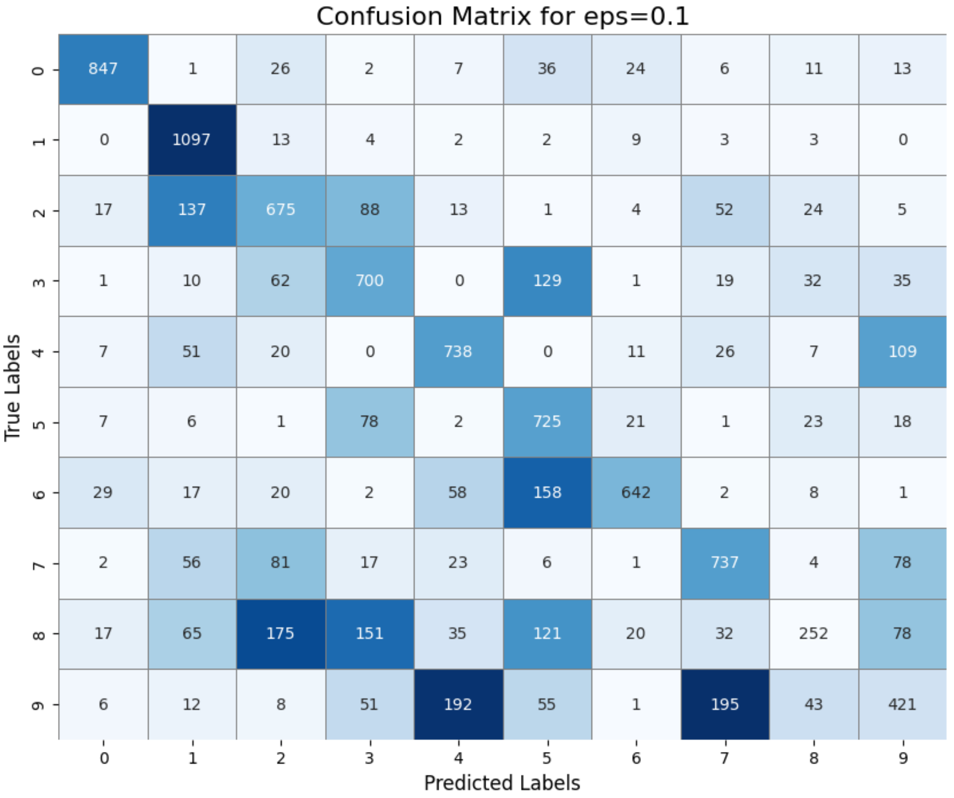


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
```

```
Accuracy on adversarial examples: 69.49%
```

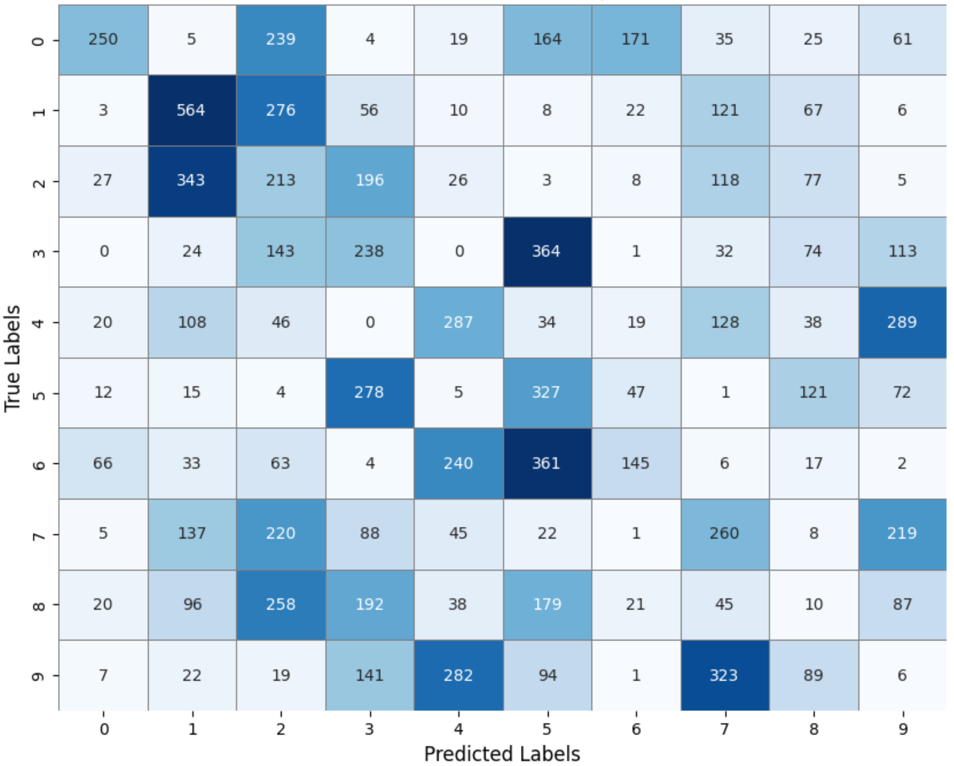


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.1
Correctly classified: 6834
Incorrectly classified: 3000
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```



```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
Accuracy on adversarial examples: 23.39%
```

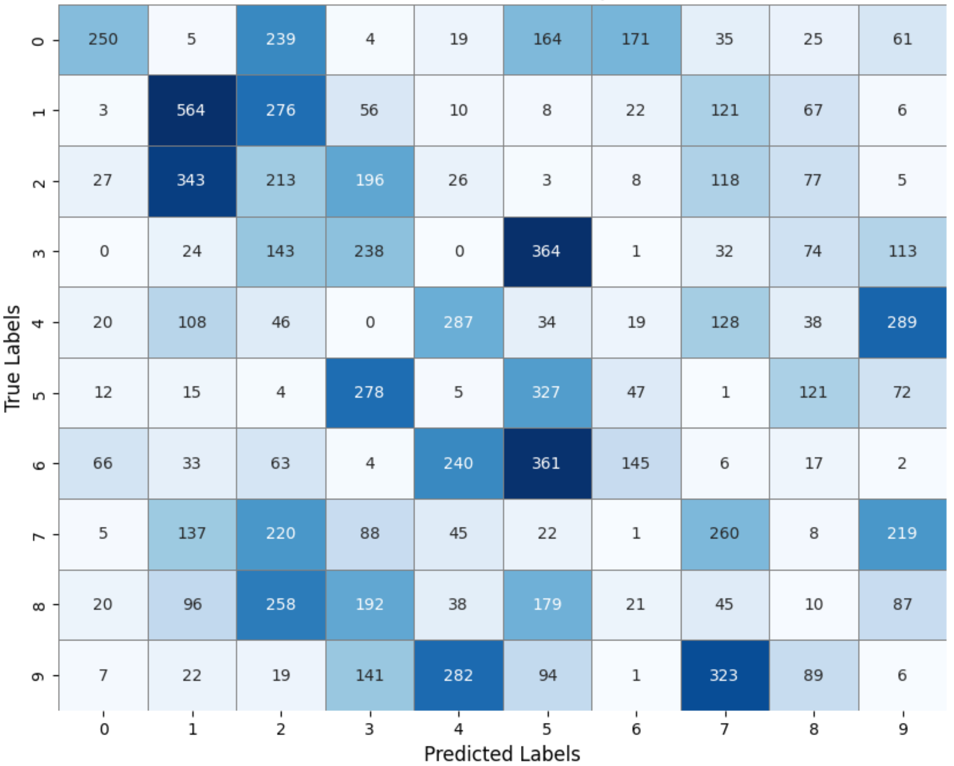
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

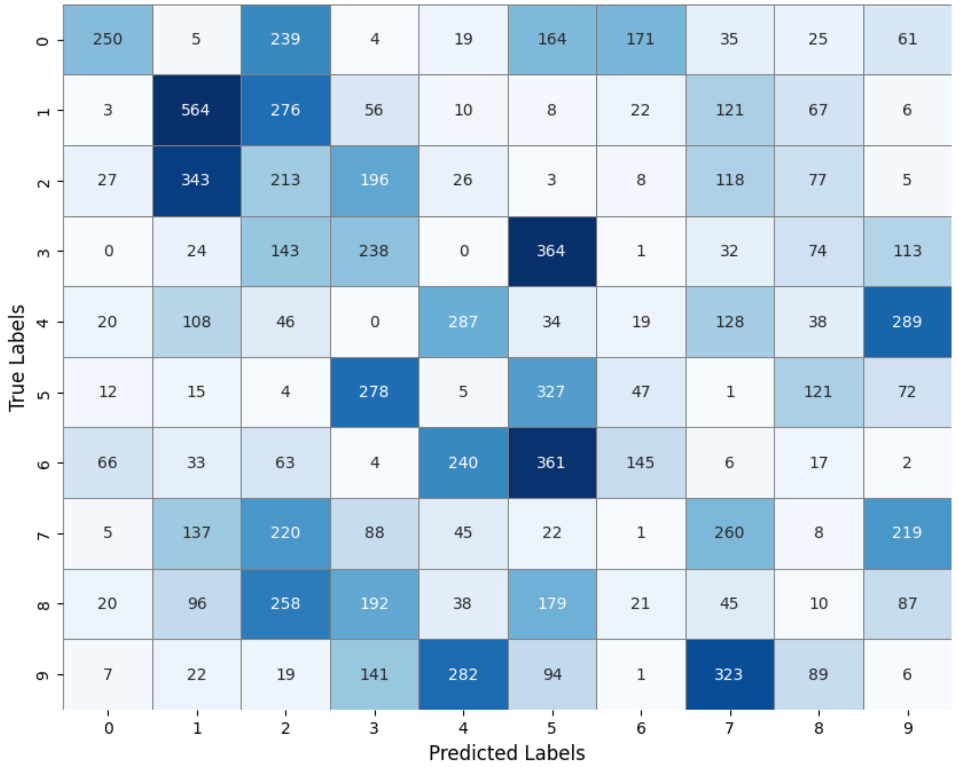
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

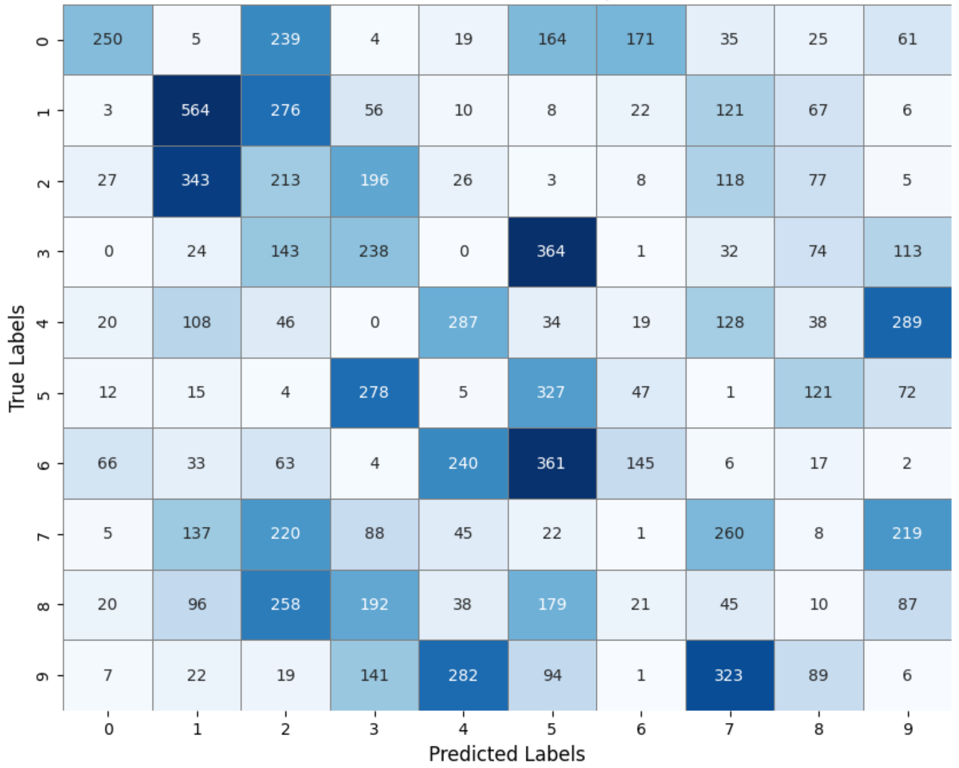
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

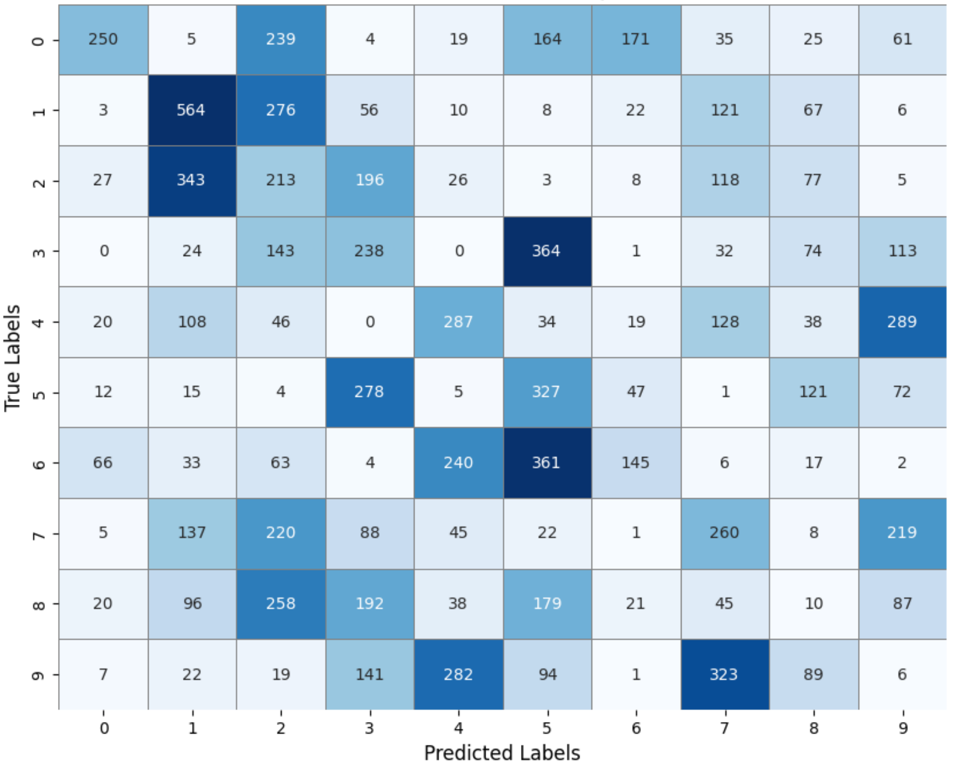
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.2
Correctly classified: 2300
Incorrectly classified: 7534
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```

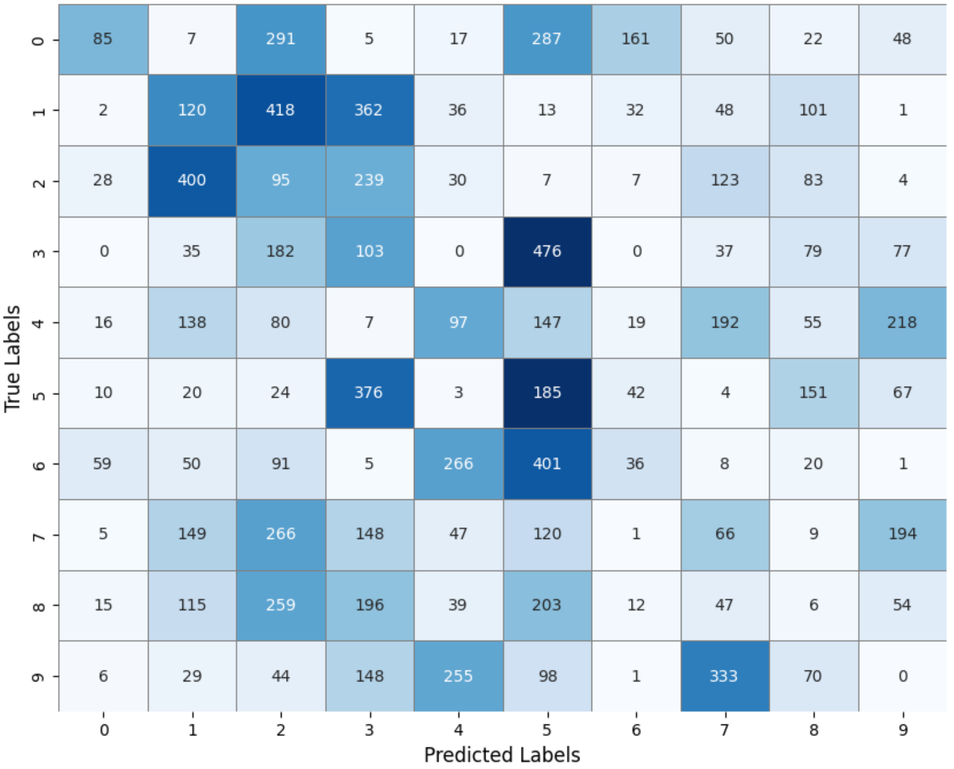
Confusion Matrix for eps=0.2



	0	1	2	3	4	5	6	7	8	9
0	250	5	239	4	19	164	171	35	25	61
1	3	564	276	56	10	8	22	121	67	6
2	27	343	213	196	26	3	8	118	77	5
3	0	24	143	238	0	364	1	32	74	113
4	20	108	46	0	287	34	19	128	38	289
5	12	15	4	278	5	327	47	1	121	72
6	66	33	63	4	240	361	145	6	17	2
7	5	137	220	88	45	22	1	260	8	219
8	20	96	258	192	38	179	21	45	10	87
9	7	22	19	141	282	94	1	323	89	6

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

Confusion Matrix for eps=0.3

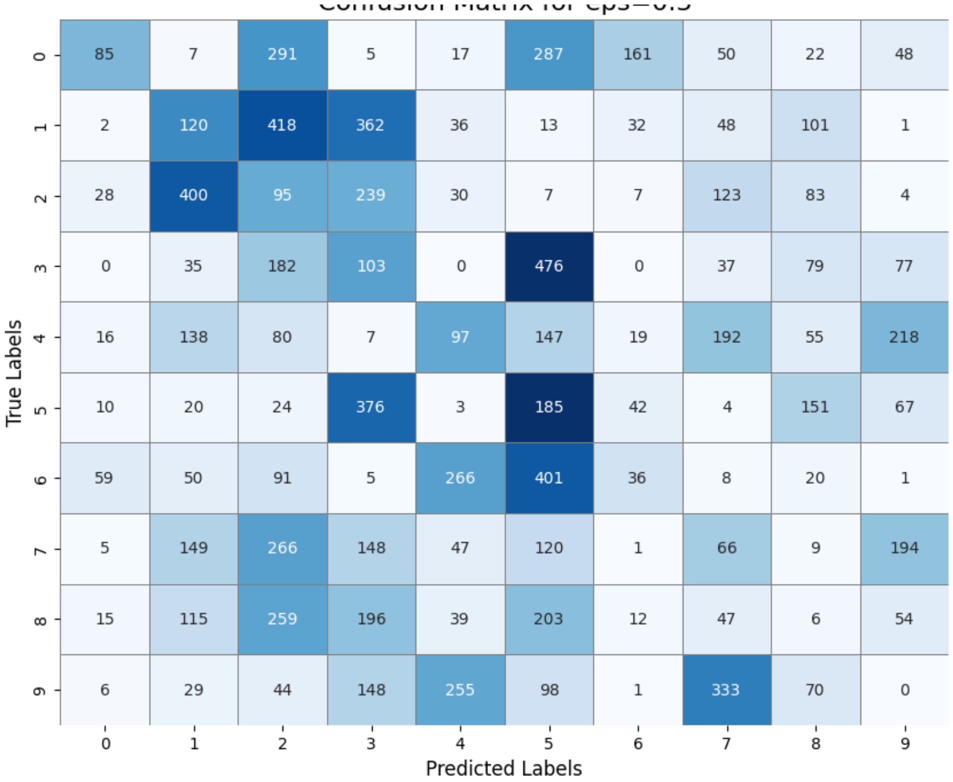


	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

Confusion Matrix for eps=0.3

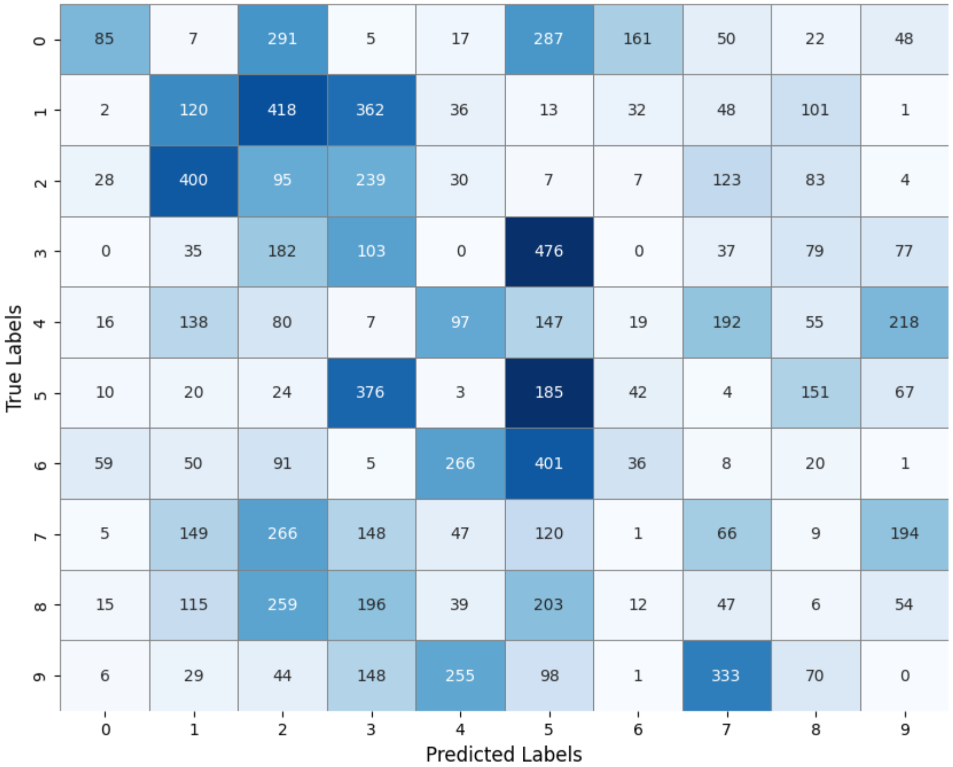
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

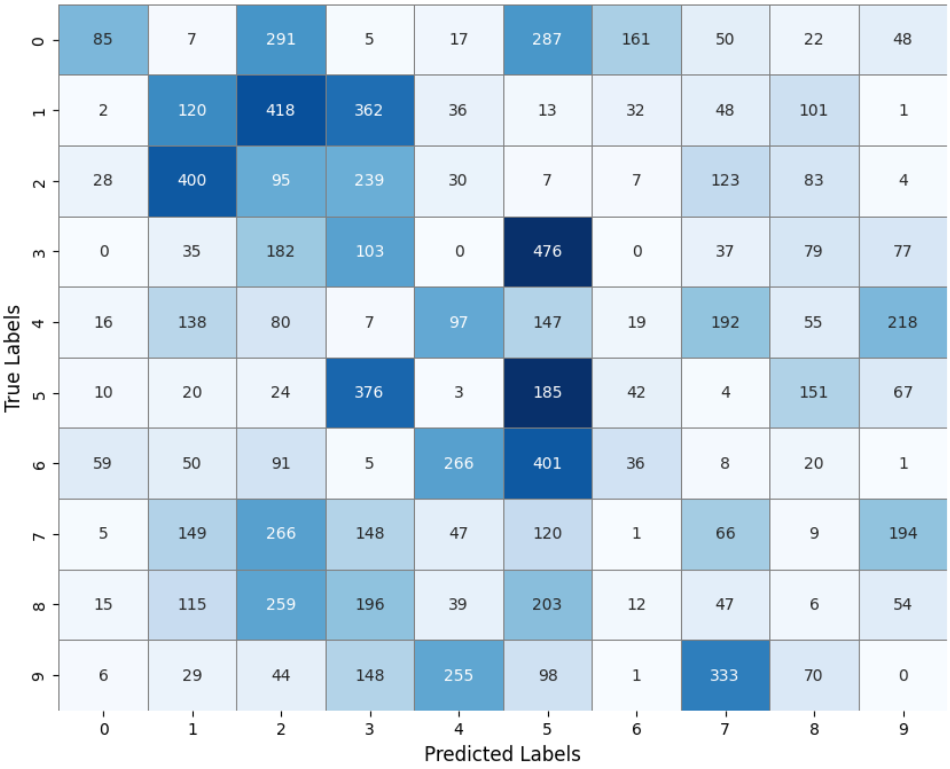
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

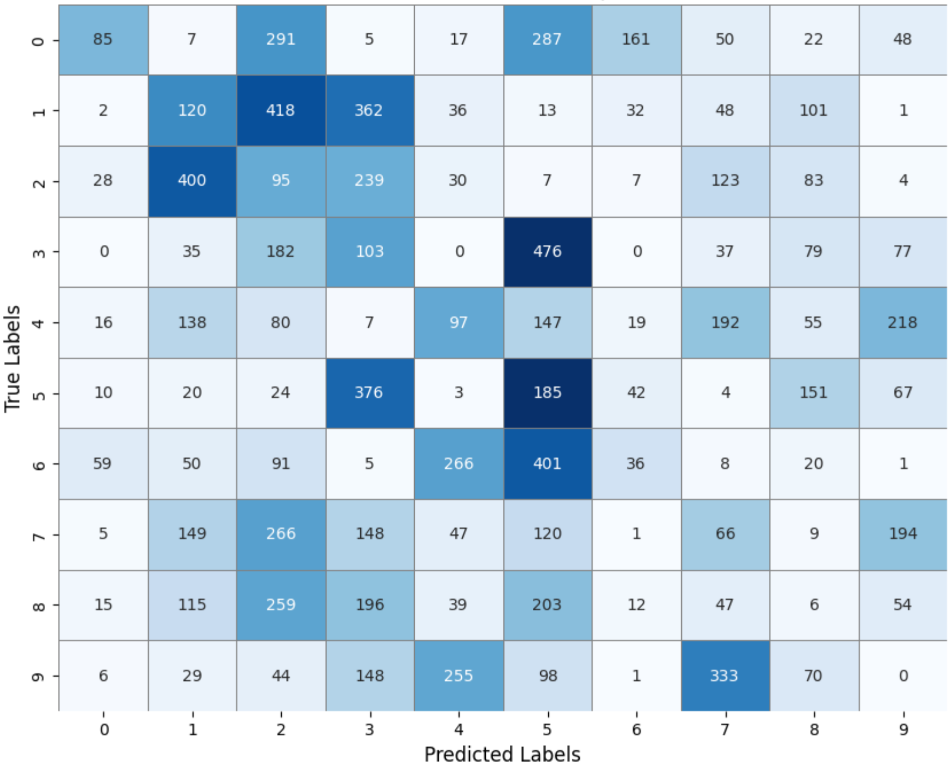
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.3
Correctly classified: 793
Incorrectly classified: 9041
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.06%
```

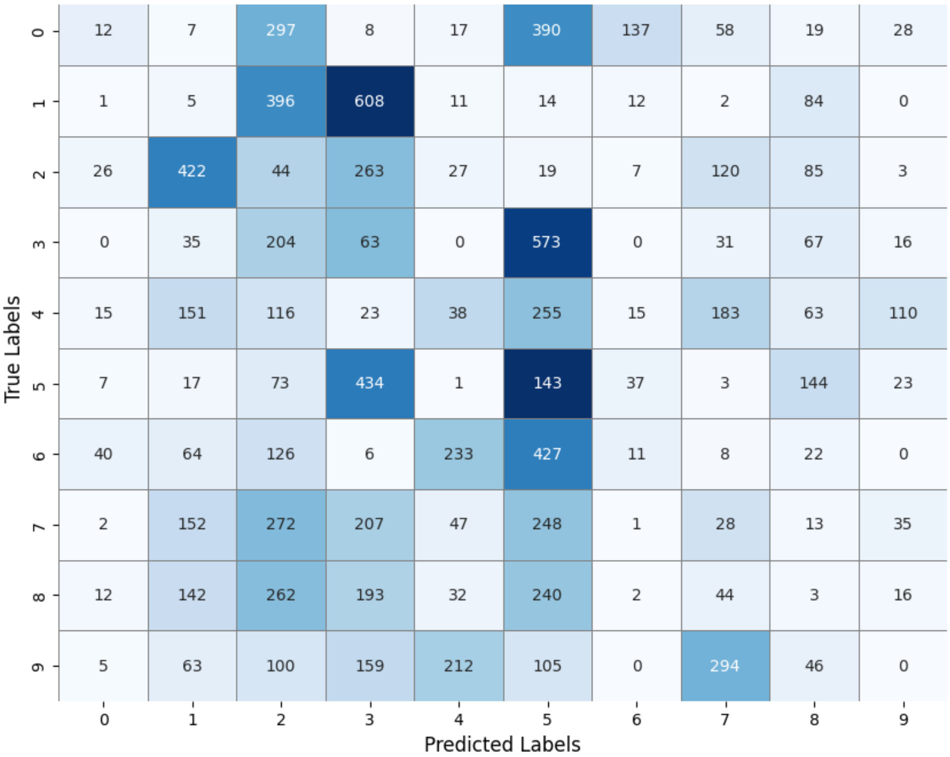
Confusion Matrix for eps=0.3



	0	1	2	3	4	5	6	7	8	9
0	85	7	291	5	17	287	161	50	22	48
1	2	120	418	362	36	13	32	48	101	1
2	28	400	95	239	30	7	7	123	83	4
3	0	35	182	103	0	476	0	37	79	77
4	16	138	80	7	97	147	19	192	55	218
5	10	20	24	376	3	185	42	4	151	67
6	59	50	91	5	266	401	36	8	20	1
7	5	149	266	148	47	120	1	66	9	194
8	15	115	259	196	39	203	12	47	6	54
9	6	29	44	148	255	98	1	333	70	0

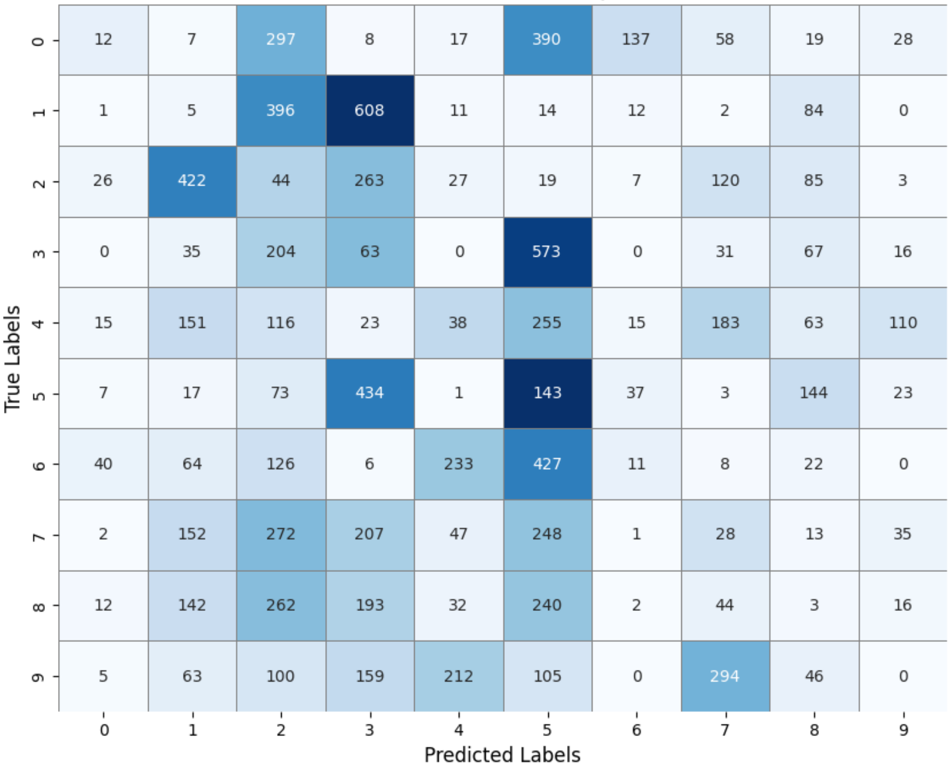
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

Confusion Matrix for eps=0.4



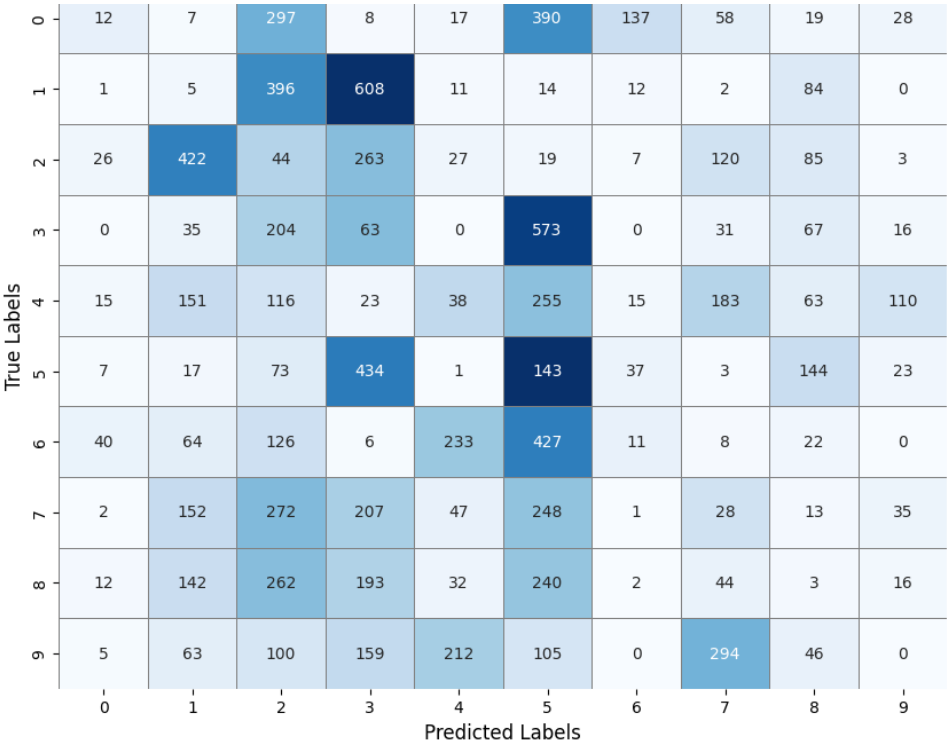
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

Confusion Matrix for eps=0.4



```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

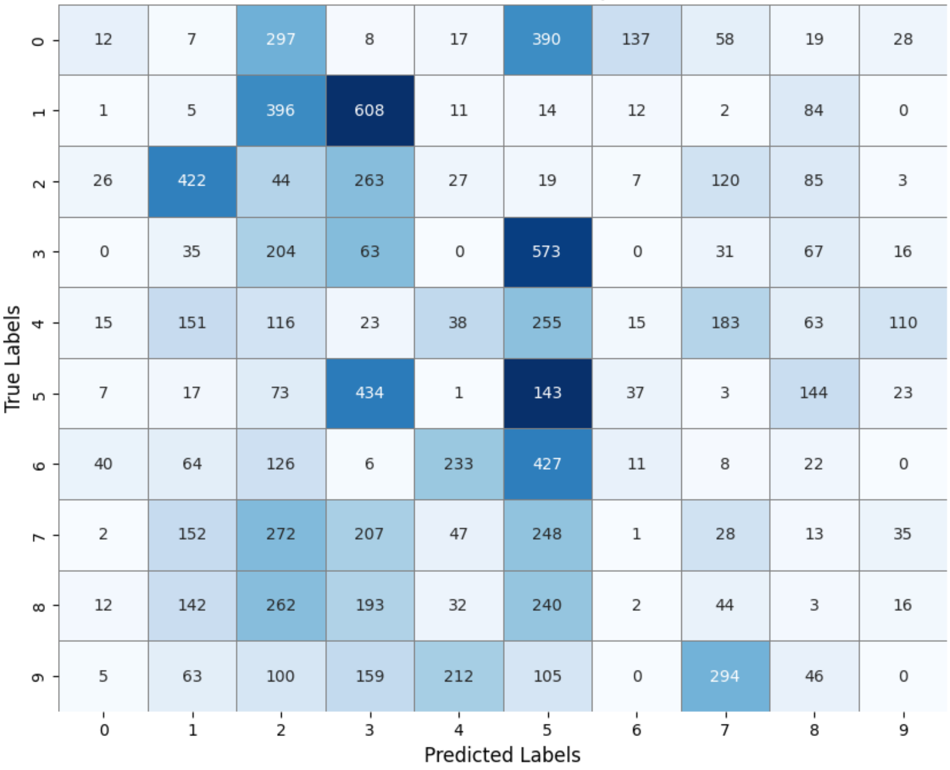
Confusion Matrix for eps=0.4



	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

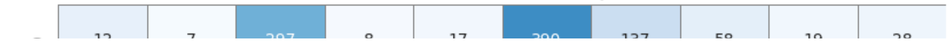
Confusion Matrix for eps=0.4

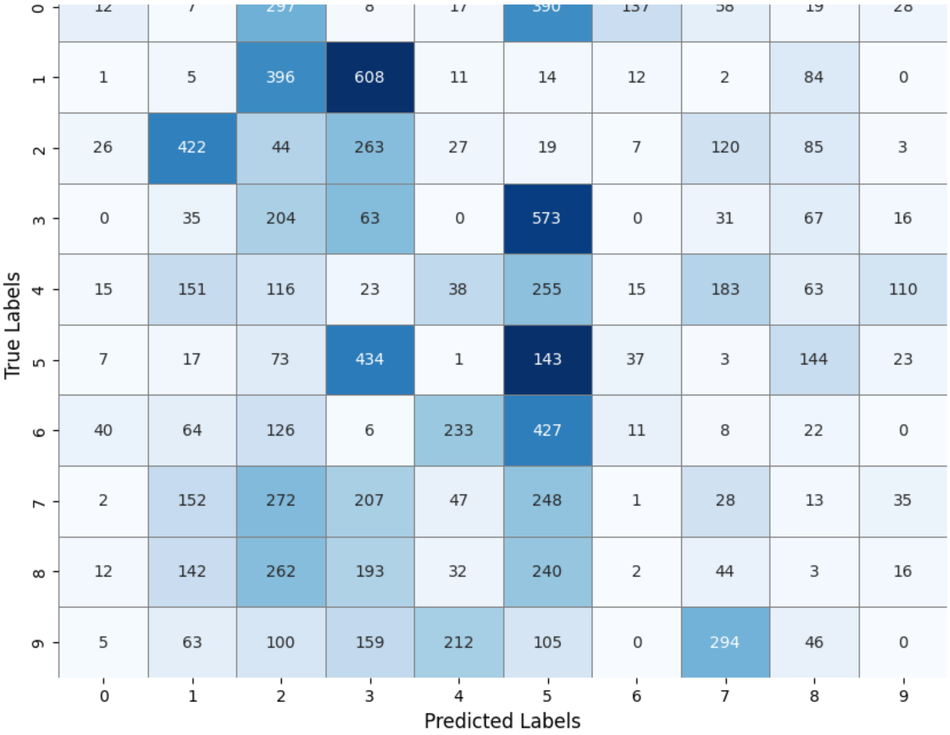


	0	1	2	3	4	5	6	7	8	9
0	12	7	297	8	17	390	137	58	19	28
1	1	5	396	608	11	14	12	2	84	0
2	26	422	44	263	27	19	7	120	85	3
3	0	35	204	63	0	573	0	31	67	16
4	15	151	116	23	38	255	15	183	63	110
5	7	17	73	434	1	143	37	3	144	23
6	40	64	126	6	233	427	11	8	22	0
7	2	152	272	207	47	248	1	28	13	35
8	12	142	262	193	32	240	2	44	3	16
9	5	63	100	159	212	105	0	294	46	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.4
Correctly classified: 347
Incorrectly classified: 9487
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

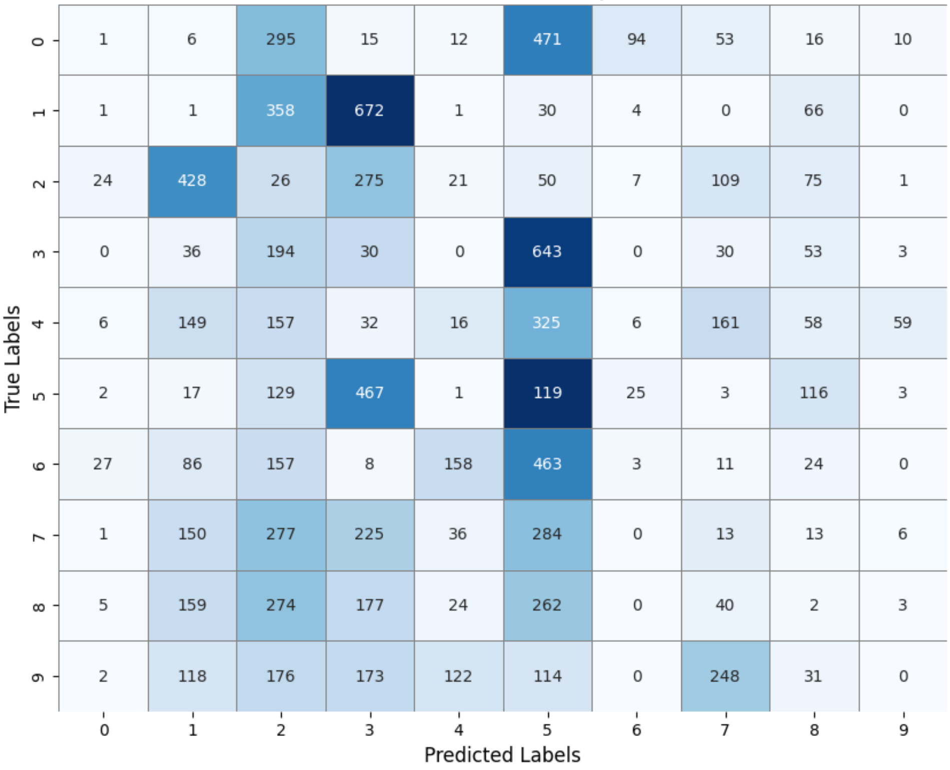
Confusion Matrix for eps=0.4





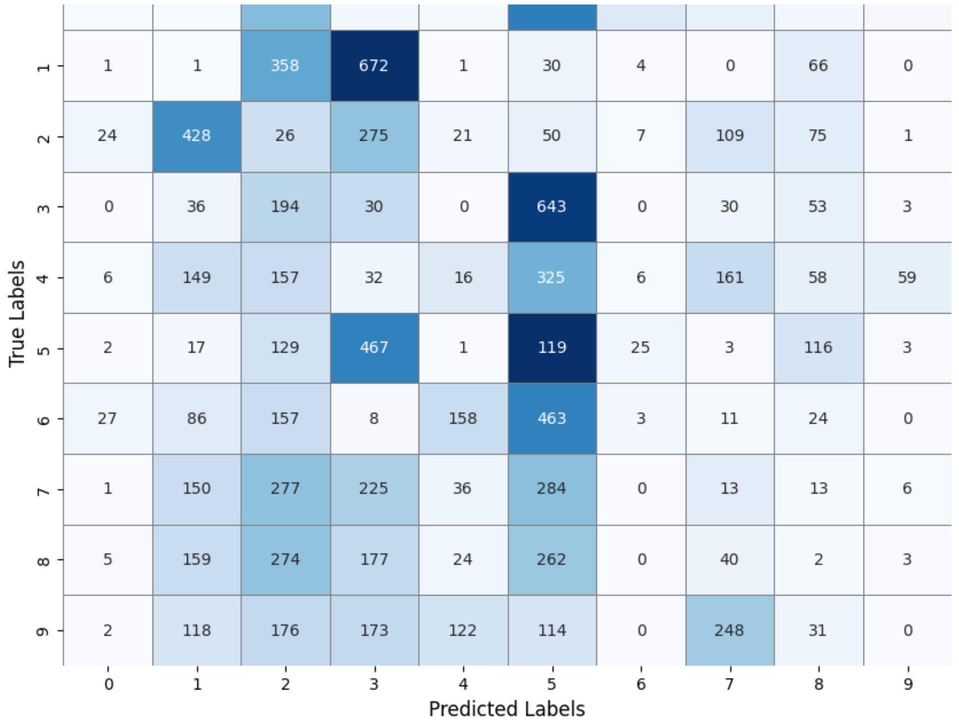
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

Confusion Matrix for eps=0.5



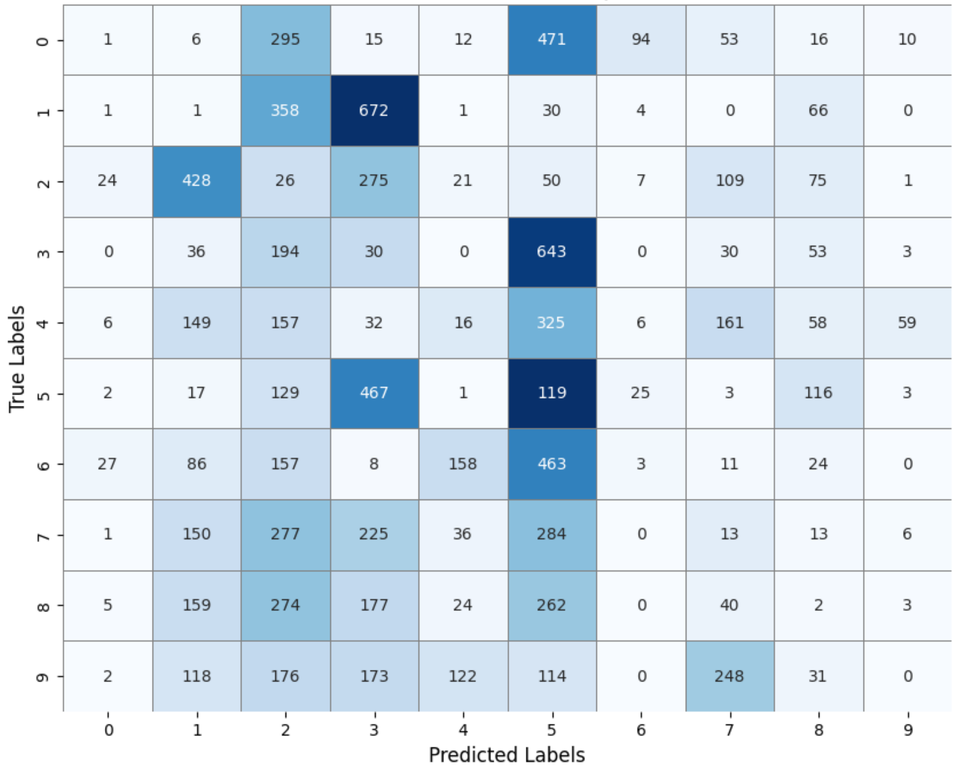
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

Confusion Matrix for eps=0.5



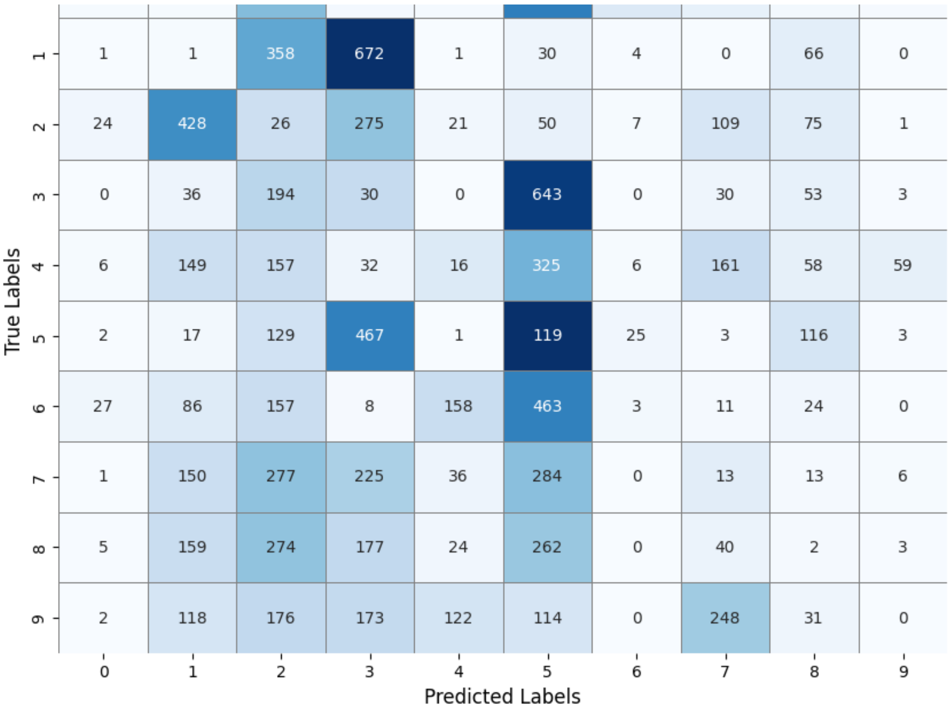
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

Confusion Matrix for eps=0.5



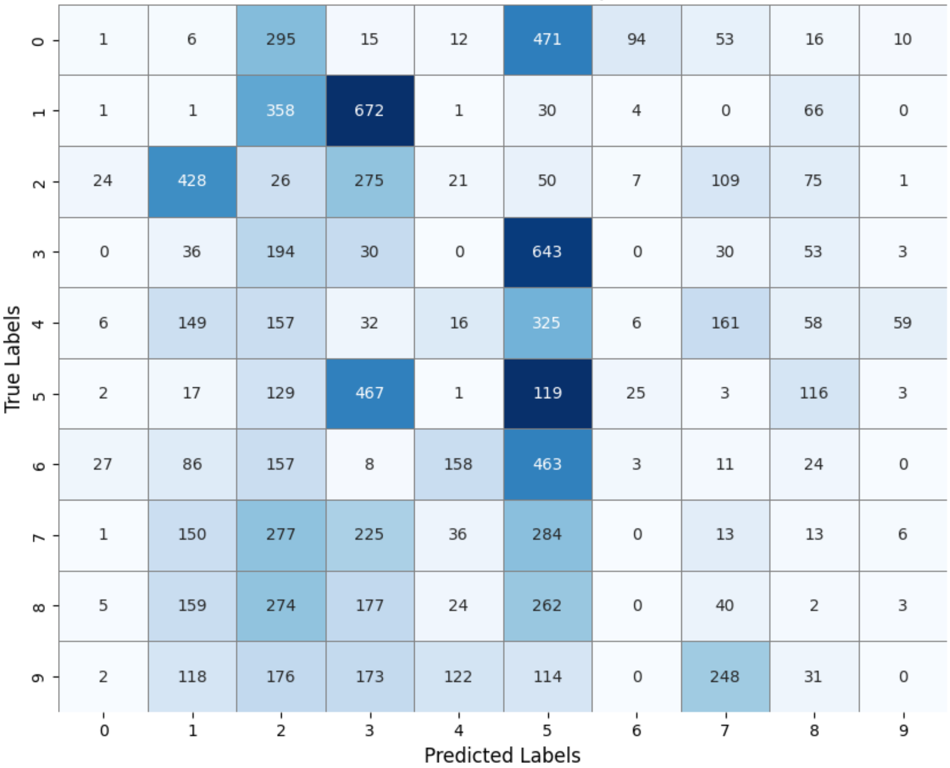
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

Confusion Matrix for eps=0.5



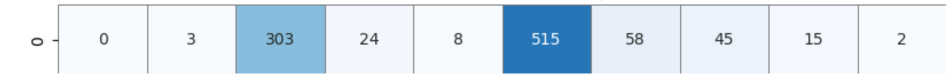
```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.5
Correctly classified: 211
Incorrectly classified: 9623
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.15%
```

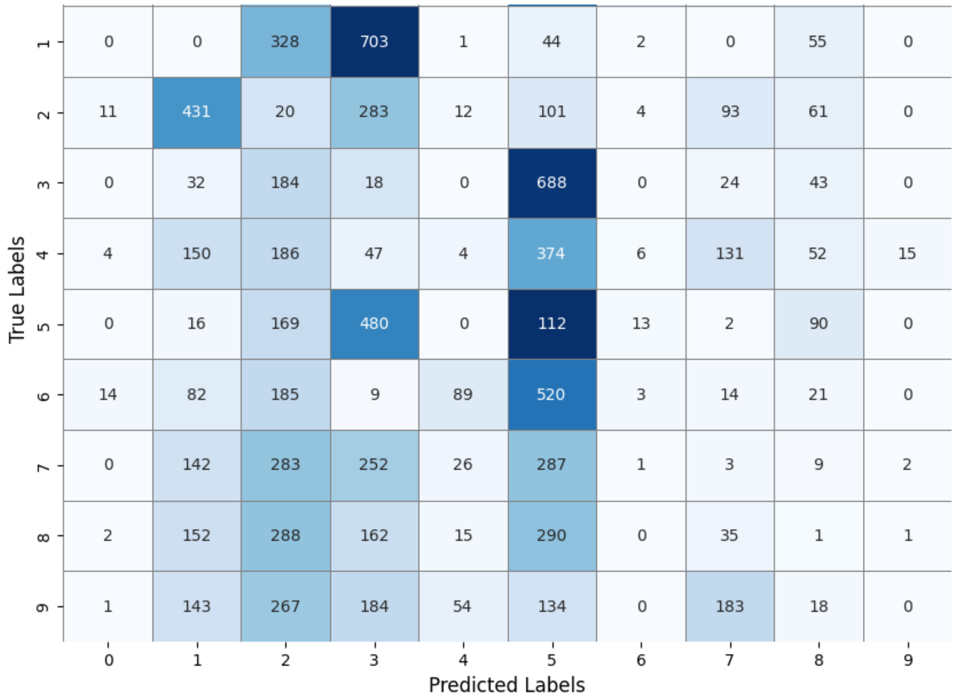
Confusion Matrix for eps=0.5



```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

Confusion Matrix for eps=0.6

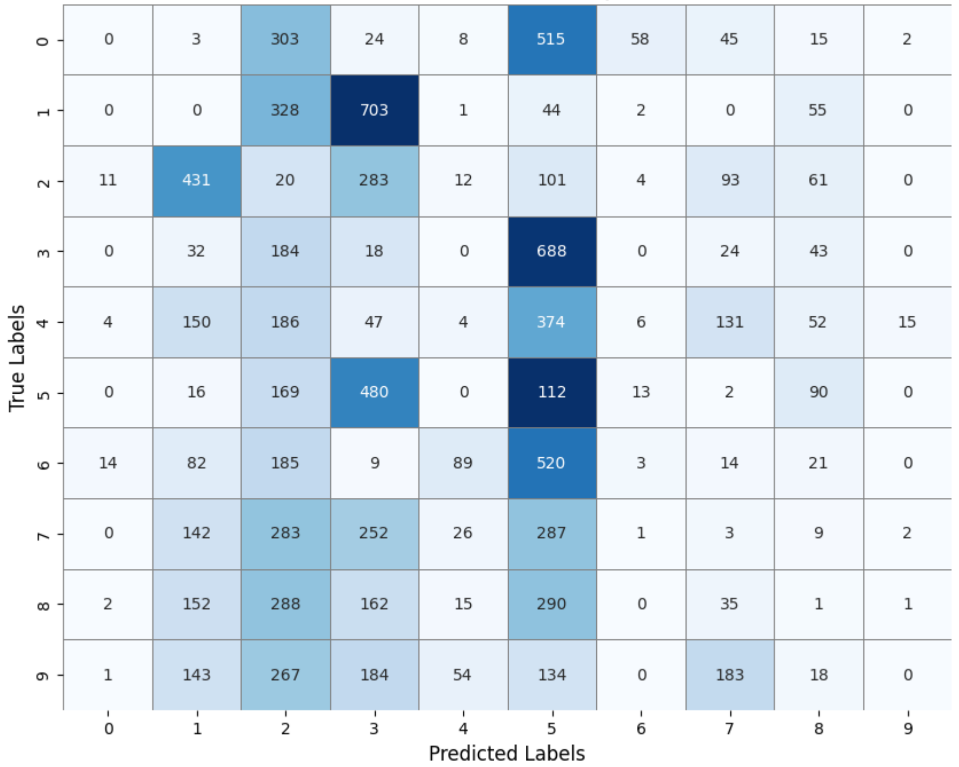




	0	1	2	3	4	5	6	7	8	9
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

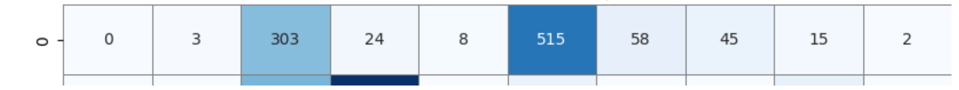
Confusion Matrix for eps=0.6



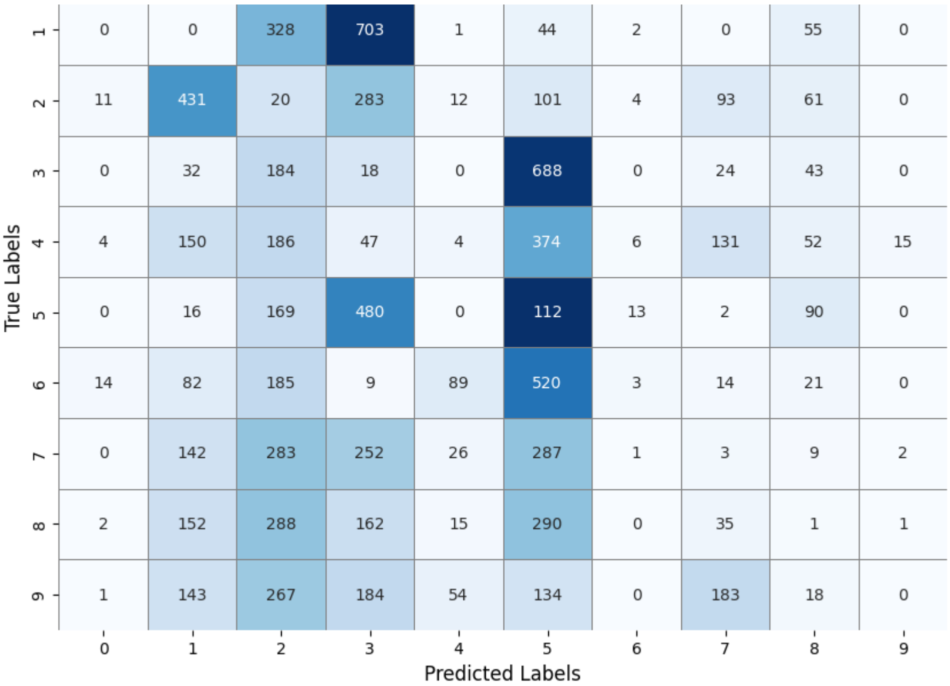
	0	1	2	3	4	5	6	7	8	9
0	0	3	303	24	8	515	58	45	15	2
1	0	0	328	703	1	44	2	0	55	0
2	11	431	20	283	12	101	4	93	61	0
3	0	32	184	18	0	688	0	24	43	0
4	4	150	186	47	4	374	6	131	52	15
5	0	16	169	480	0	112	13	2	90	0
6	14	82	185	9	89	520	3	14	21	0
7	0	142	283	252	26	287	1	3	9	2
8	2	152	288	162	15	290	0	35	1	1
9	1	143	267	184	54	134	0	183	18	0

```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```

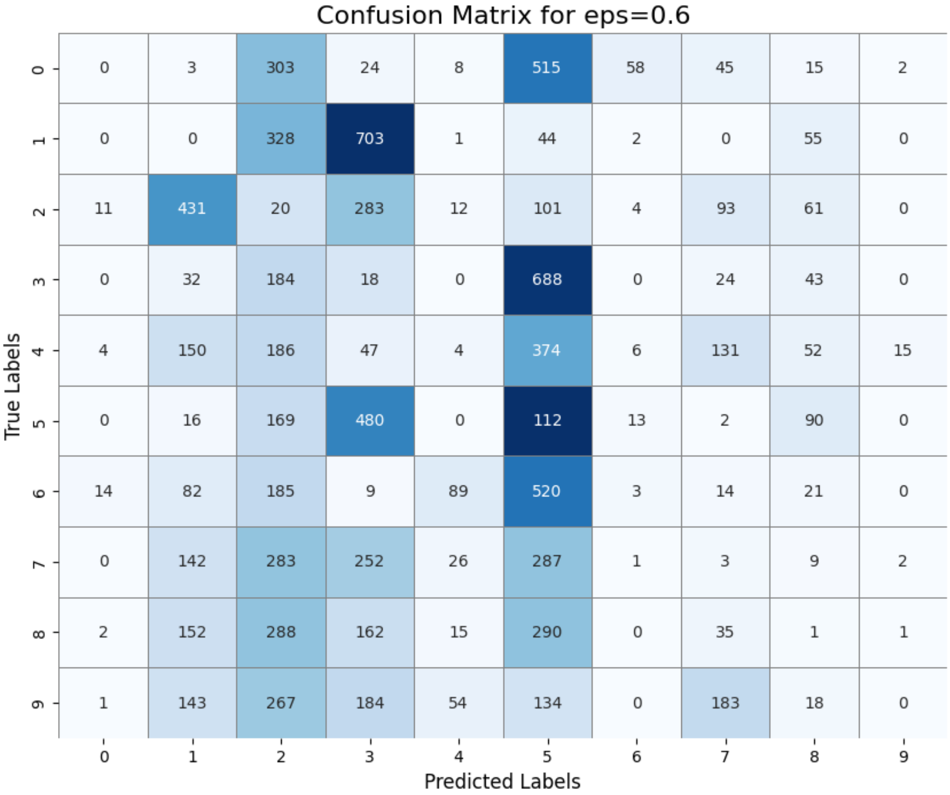
Confusion Matrix for eps=0.6



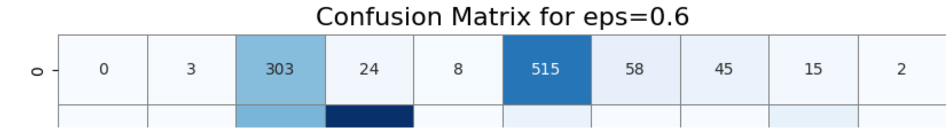
0	0	3	303	24	8	515	58	45	15	2

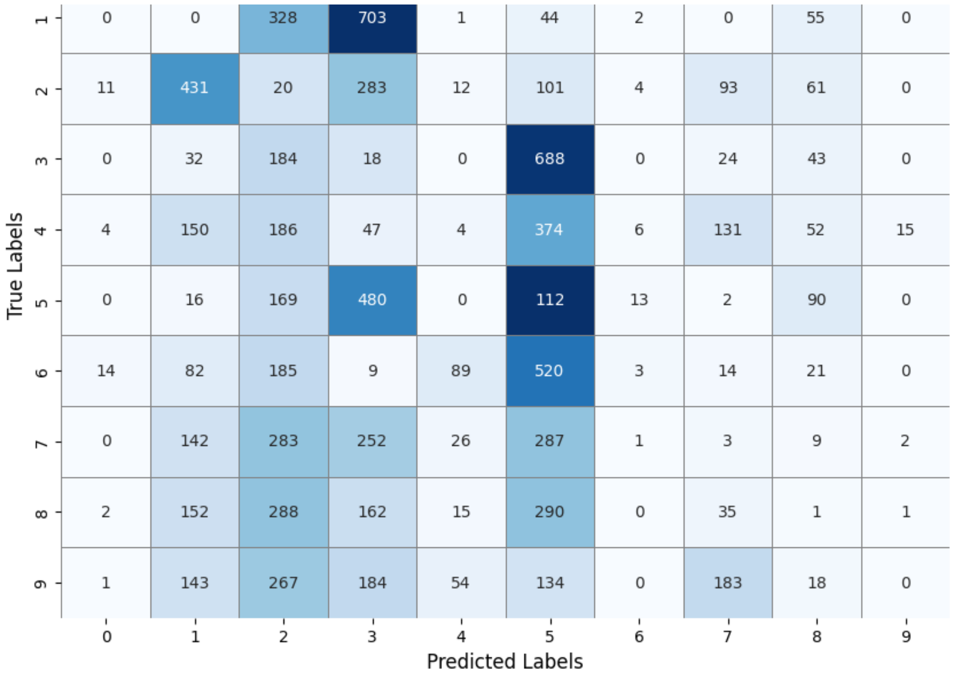


```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```



```
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
Adversarial test data: eps:0.6
Correctly classified: 161
Incorrectly classified: 9673
accuracy_clean:1.0
Accuracy on clean data: 100.00%
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.64%
```





```
     eps  attack_num  total_correct  total_adv  \
0   0.01           1           9834       9834
1   0.01           2           9834       9834
2   0.01           3           9834       9834
3   0.01           4           9834       9834
4   0.01           5           9834       9834
5   0.02           1           9834       9834
6   0.02           2           9834       9834
7   0.02           3           9834       9834
8   0.02           4           9834       9834
9   0.02           5           9834       9834
10  0.03           1           9834       9834
11  0.03           2           9834       9834
12  0.03           3           9834       9834
13  0.03           4           9834       9834
14  0.03           5           9834       9834
15  0.04           1           9834       9834
16  0.04           2           9834       9834
17  0.04           3           9834       9834
18  0.04           4           9834       9834
19  0.04           5           9834       9834
20  0.05           1           9834       9834
21  0.05           2           9834       9834
22  0.05           3           9834       9834
23  0.05           4           9834       9834
24  0.05           5           9834       9834
25  0.10           1           9834       9834
26  0.10           2           9834       9834
27  0.10           3           9834       9834
28  0.10           4           9834       9834
29  0.10           5           9834       9834
30  0.20           1           9834       9834
31  0.20           2           9834       9834
32  0.20           3           9834       9834
33  0.20           4           9834       9834
34  0.20           5           9834       9834
35  0.30           1           9834       9834
36  0.30           2           9834       9834
37  0.30           3           9834       9834
38  0.30           4           9834       9834
39  0.30           5           9834       9834
40  0.40           1           9834       9834
41  0.40           2           9834       9834
42  0.40           3           9834       9834
43  0.40           4           9834       9834
44  0.40           5           9834       9834
45  0.50           1           9834       9834
46  0.50           2           9834       9834
47  0.50           3           9834       9834
48  0.50           4           9834       9834
49  0.50           5           9834       9834
50  0.60           1           9834       9834
51  0.60           2           9834       9834
52  0.60           3           9834       9834
53  0.60           4           9834       9834
```

```
54  0.60           5           9834       9834

                                    correct_adv_counts
0   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
1   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
2   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
3   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
4   {0: 971, 1: 1141, 2: 1017, 3: 990, 4: 971, 5: ...
5   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
6   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
7   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
8   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
9   {0: 975, 1: 1156, 2: 1022, 3: 990, 4: 972, 5: ...
10  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
11  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
12  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
13  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
14  {0: 984, 1: 1177, 2: 1011, 3: 996, 4: 971, 5: ...
15  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
16  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
17  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
18  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
19  {0: 986, 1: 1191, 2: 1024, 3: 992, 4: 977, 5: ...
20  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
21  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
22  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
23  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
24  {0: 986, 1: 1227, 2: 1030, 3: 998, 4: 1000, 5:...
25  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
26  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
27  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
28  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
29  {0: 933, 1: 1452, 2: 1081, 3: 1093, 4: 1070, 5...
30  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
31  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
32  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
33  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
34  {0: 410, 1: 1347, 2: 1481, 3: 1197, 4: 952, 5:...
35  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
36  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
37  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
38  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
39  {0: 226, 1: 1063, 2: 1750, 3: 1589, 4: 790, 5:...
40  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
41  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
42  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
43  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
44  {0: 120, 1: 1058, 2: 1890, 3: 1964, 4: 618, 5:...
45  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
46  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
47  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
48  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
49  {0: 69, 1: 1150, 2: 2043, 3: 2074, 4: 391, 5: ...
50  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
51  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
52  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
53  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
54  {0: 32, 1: 1151, 2: 2213, 3: 2162, 4: 209, 5: ...
<ipython-input-66-119194b068fb>:66: FutureWarning: The frame.append method is
  results_df = results_df.append({
```