# Intal FoolBox Library

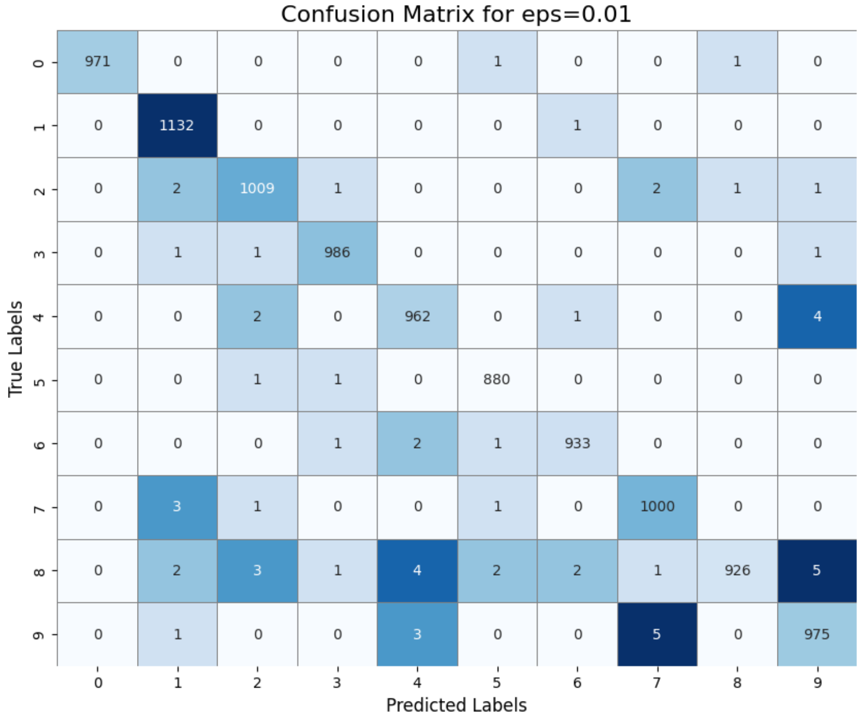
```
!pip install foolbox
```

# Fast Gradient Sign Method (FGSM)

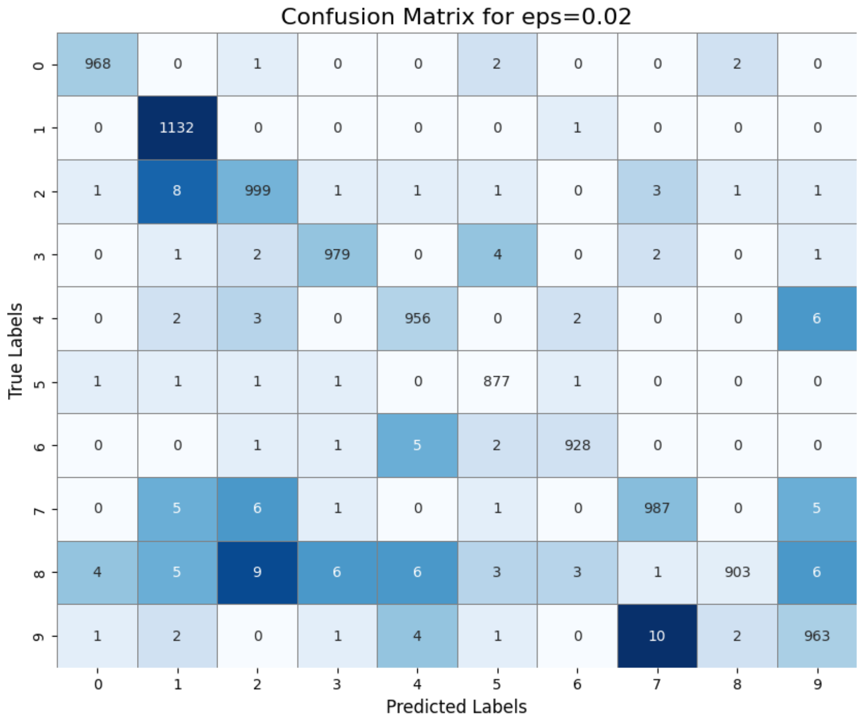
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
import foolbox as fb
import eagerpy as ep
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

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

# Load and preprocess the examples
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy').astype('int32')  # Cast labels to int32

# Create a Foolbox model
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))



# Calculate accuracy on clean data
clean_predictions = model.predict(correct_examples).argmax(axis=-1)
accuracy_clean = np.mean(clean_predictions == correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Convert data to TensorFlow tensors
images = tf.convert_to_tensor(correct_examples)
labels = tf.convert_to_tensor(correct_labels, dtype=tf.int32)  # Cast labels to int32

# Create a Foolbox model for TensorFlow
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

# Apply an FGSM attack
attack = fb.attacks.FGSM()
epsilons = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
raw_advs, clipped_advs, success = attack(fmodel, images, labels, epsilons=epsilons)




# Assuming 'clipped_advs' are the adversarial examples for different epsilons
for eps, advs_ in zip(epsilons, clipped_advs):
    # Predict the labels of the adversarial examples
    y_adv = np.argmax(model.predict(advs_), axis=1)

    # Calculate accuracy on adversarial examples
    accuracy_adv = np.mean(y_adv == correct_labels)
    print(f"\nAdversarial 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
    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
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey')

    # 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)

    # Save the plot
    image_filename = f'confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()  # Display the figure in the notebook
```

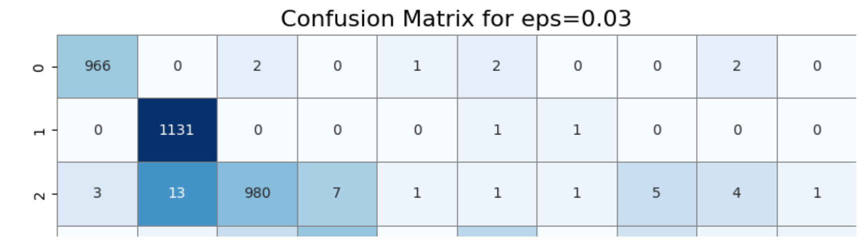
```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/foolbox/model
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
308/308 [==============================] - 8s 2ms/step
Accuracy on clean data: 100.00%
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.39%
```

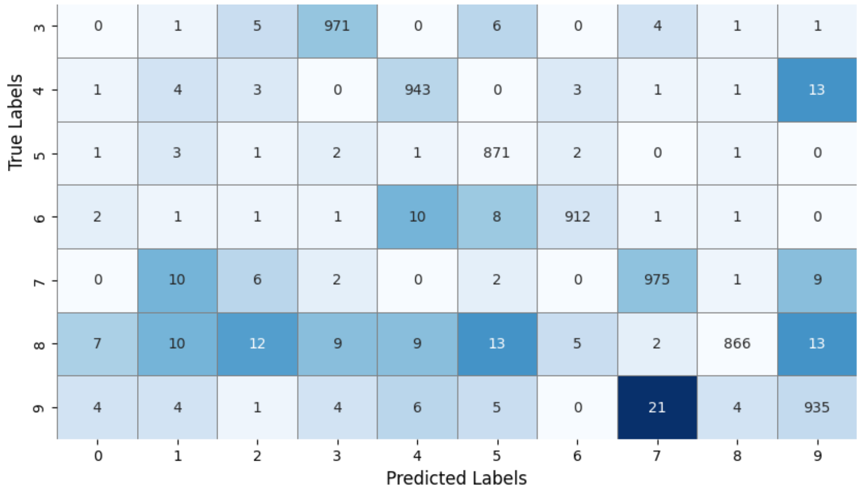



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.56%
```



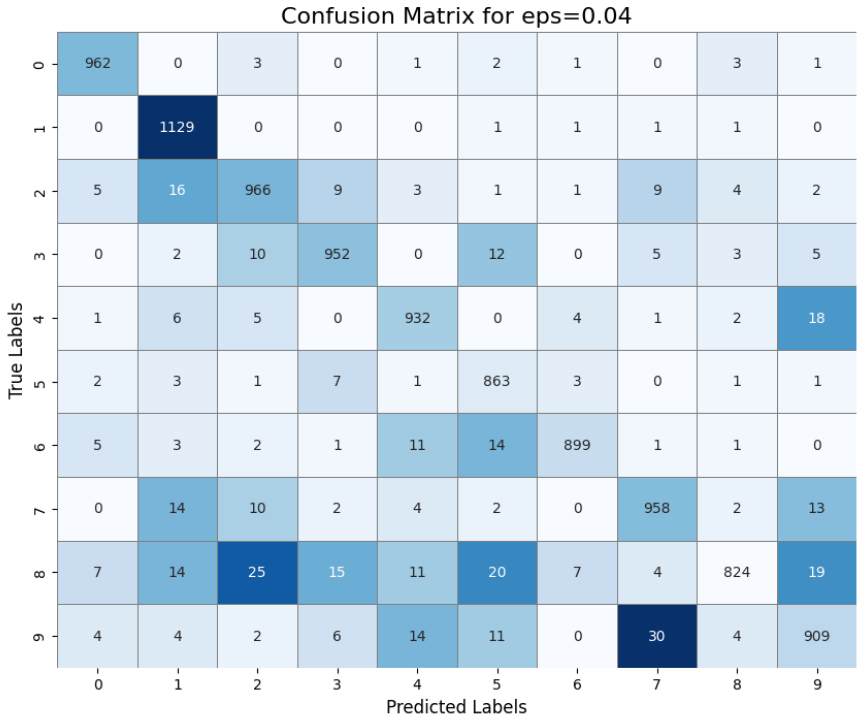

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.11%
```





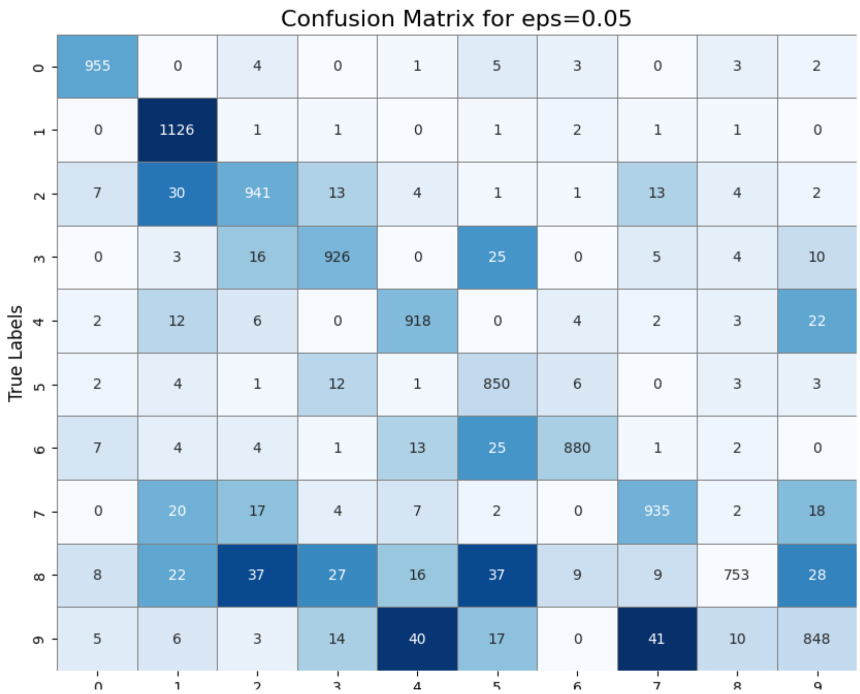
308/308 [==============================] - 1s 2ms/step

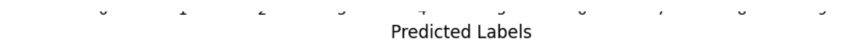
Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.53%



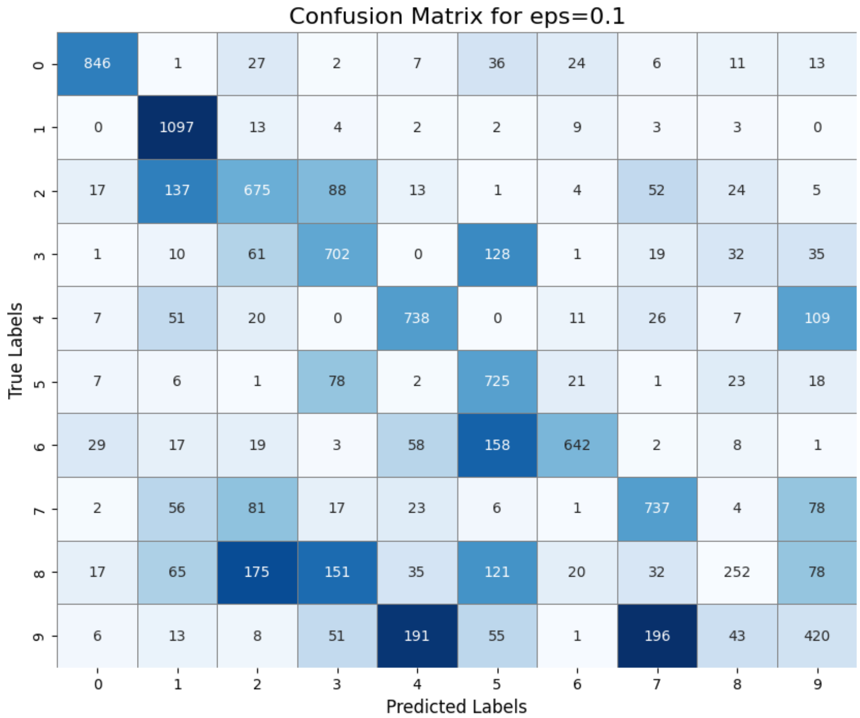
308/308 [==============================] - 1s 2ms/step

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

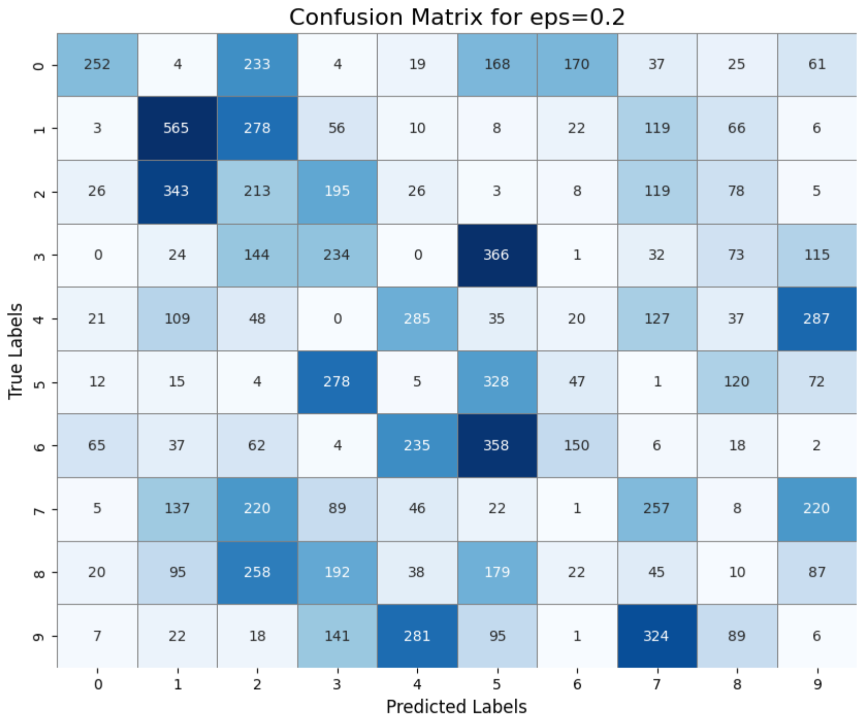




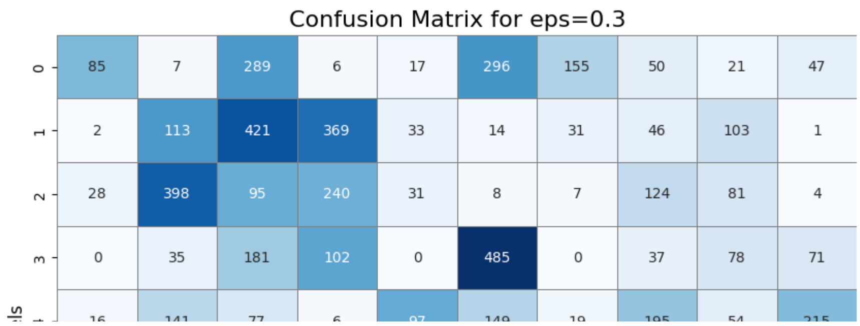

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.49%
```

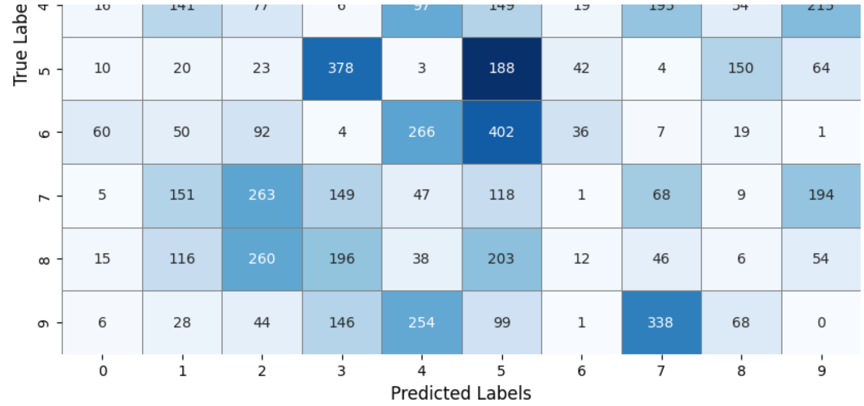



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 23.39%
```



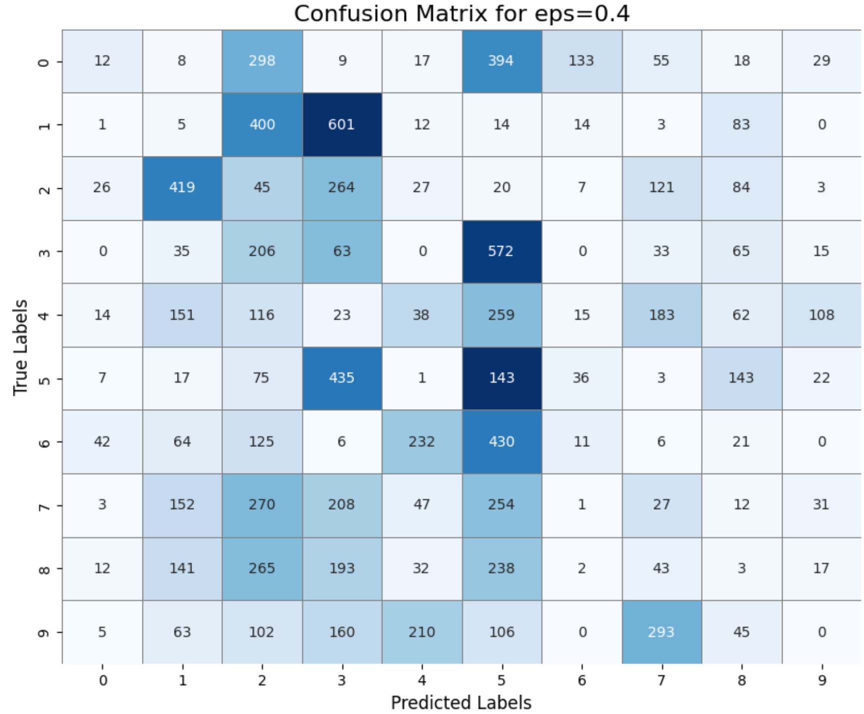

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 8.03%
```





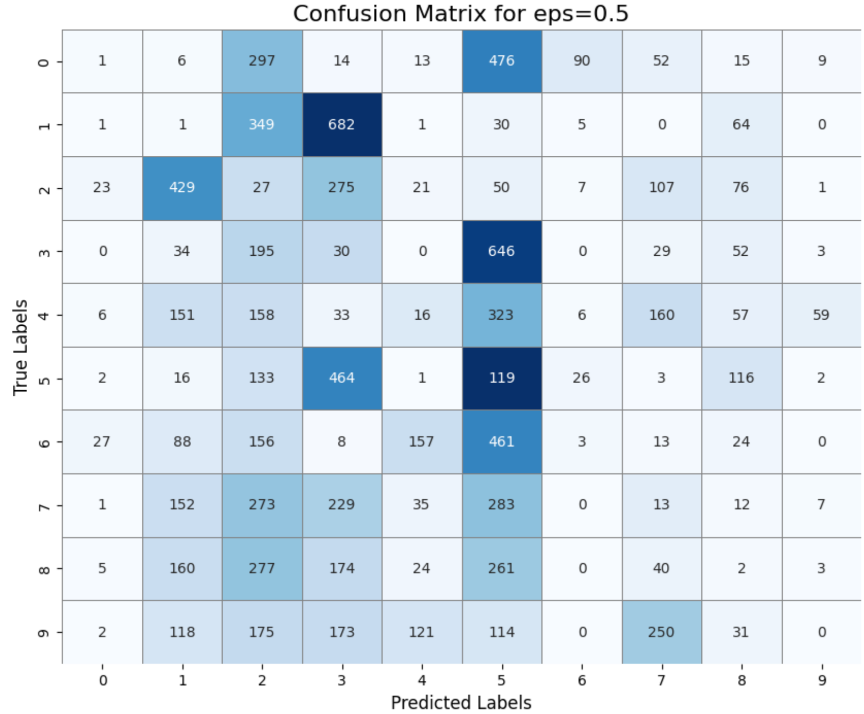

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 3.53%
```

Confusion Matrix for eps=0.4



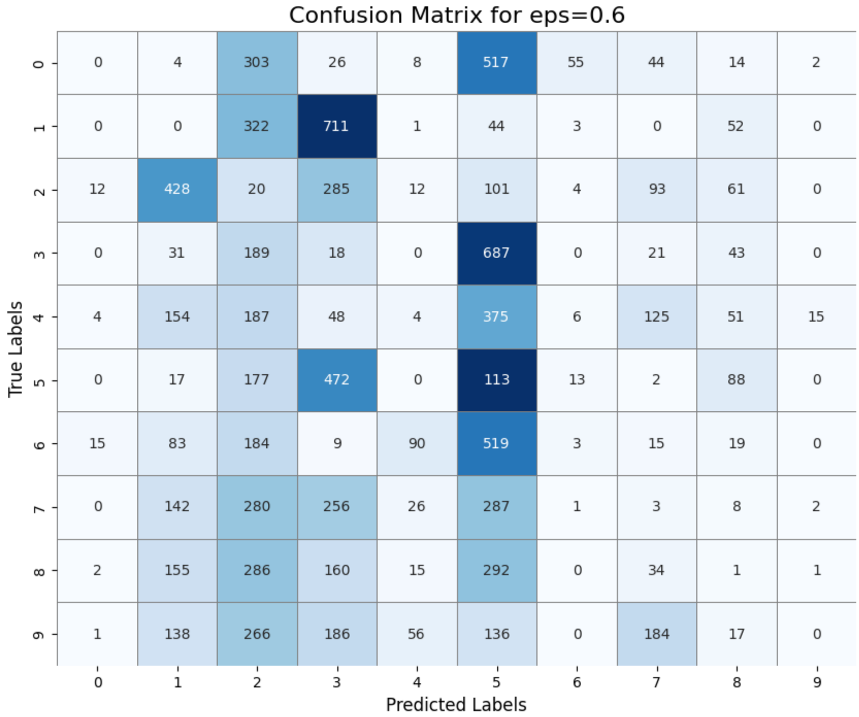

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 2.16%
```

Confusion Matrix for eps=0.5




```
308/308 [==============================] - 1s 3ms/step
```

```
Adversarial test data: eps:0.6
Accuracy on adversarial examples: 1.65%
```




```
correct_examples.shape
```

```
(9834, 28, 28, 1)
```

```
images.shape
```

```
TensorShape([9834, 28, 28, 1])
```

```
correct_labels.shape
```

```
(9834,)
```

```
labels.shape
```

```
TensorShape([9834])
```

```
robust_accuracy
```

```
<tf.Tensor: shape=(11,), dtype=float32, numpy=
array([0.9938987 , 0.9855603 , 0.9711206 , 0.9552573 , 0.92861503,
       0.6951393 , 0.17937768, 0.02379501, 0.0043726 , 0.00274557,
       0.0025422 ], dtype=float32)>
```

## Calculate and report the robust accuracy

```
# Calculate and report the robust accuracy
# Convert 'success' to float, and calculate the mean
robust_accuracy = 1 - tf.reduce_mean(tf.cast(success, tf.float32), axis=-1)
print("robust accuracy for perturbations with")
for eps, acc in zip(epsilons, robust_accuracy):
    print(f"  Linf norm ≤ {eps:<6}: {acc.numpy() * 100:4.1f} %")
```

```
robust accuracy for perturbations with
  Linf norm ≤ 0.01  : 99.4 %
  Linf norm ≤ 0.02  : 98.6 %
  Linf norm ≤ 0.03  : 97.1 %
  Linf norm ≤ 0.04  : 95.5 %
  Linf norm ≤ 0.05  : 92.9 %
  Linf norm ≤ 0.1   : 69.5 %
  Linf norm ≤ 0.2   : 17.9 %
  Linf norm ≤ 0.3   :  2.4 %
  Linf norm ≤ 0.4   :  0.4 %
  Linf norm ≤ 0.5   :  0.3 %
  Linf norm ≤ 0.6   :  0.3 %
```

```
import numpy as np
import tensorflow as tf

# Assuming 'clipped_advs' and 'images' are available as tensors
for eps, advs_ in zip(epsilons, clipped_advs):
    # Flatten the spatial dimensions of the adversarial and original images
    original_flat = tf.reshape(images, [images.shape[0], -1])
    adversarial_flat = tf.reshape(advs_, [advs_.shape[0], -1])

    # Calculate Linf norm using TensorFlow
    perturbation_sizes = tf.norm(adversarial_flat - original_flat, ord=np.inf, axis=1)
    print(f"Linf norm ≤ {eps:<6}: perturbation sizes:", perturbation_sizes.numpy())
```

```
Linf norm ≤ 0.01  : perturbation sizes: [0.01000001 0.01000001 0.01000001 ... 0.01000001 0.01000002 0.01000002]
Linf norm ≤ 0.02  : perturbation sizes: [0.02000001 0.02000001 0.02000001 ... 0.02000001 0.02000001 0.02000001]
Linf norm ≤ 0.03  : perturbation sizes: [0.03 0.03 0.03 ... 0.03 0.03 0.03]
Linf norm ≤ 0.04  : perturbation sizes: [0.04000002 0.04000002 0.04000002 ... 0.04000002 0.04000002 0.04000002]
Linf norm ≤ 0.05  : perturbation sizes: [0.05000001 0.05000001 0.05000001 ... 0.05000001 0.05000001 0.05000001]
Linf norm ≤ 0.1   : perturbation sizes: [0.10000002 0.10000002 0.10000002 ... 0.10000002 0.10000002 0.10000002]
Linf norm ≤ 0.2   : perturbation sizes: [0.20000002 0.20000002 0.20000002 ... 0.20000002 0.20000002 0.20000002]
Linf norm ≤ 0.3   : perturbation sizes: [0.30000004 0.30000004 0.30000004 ... 0.30000004 0.30000004 0.30000004]
Linf norm ≤ 0.4   : perturbation sizes: [0.40000004 0.40000004 0.40000004 ... 0.40000004 0.40000004 0.40000004]
Linf norm ≤ 0.5   : perturbation sizes: [0.5 0.5 0.5 ... 0.5 0.5 0.5]
Linf norm ≤ 0.6   : perturbation sizes: [0.6 0.6 0.6 ... 0.6 0.6 0.6]
```

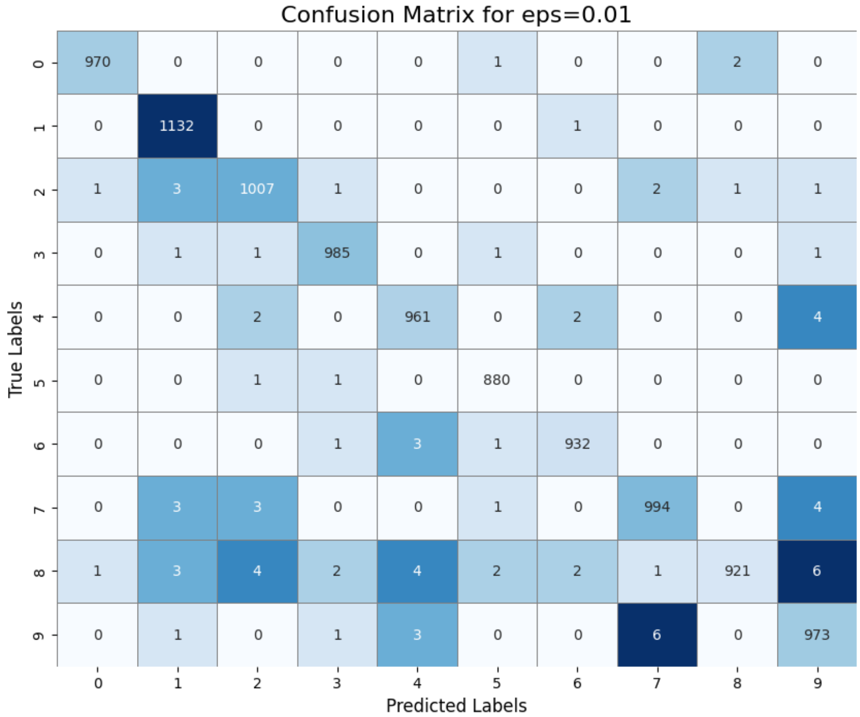
## ProjectedGradientDescentAttack

```
import numpy as np
import tensorflow as tf
from keras.models import load_model
import foolbox as fb
import eagerpy as ep
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

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

# Load and preprocess the examples
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy').astype('int32')  # Cast labels to int32

# Create a Foolbox model
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))



# Calculate accuracy on clean data
clean_predictions = model.predict(correct_examples).argmax(axis=-1)
accuracy_clean = np.mean(clean_predictions == correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Convert data to TensorFlow tensors
images = tf.convert_to_tensor(correct_examples)
labels = tf.convert_to_tensor(correct_labels, dtype=tf.int32)  # Cast labels to int32

# Create a Foolbox model for TensorFlow
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))
```

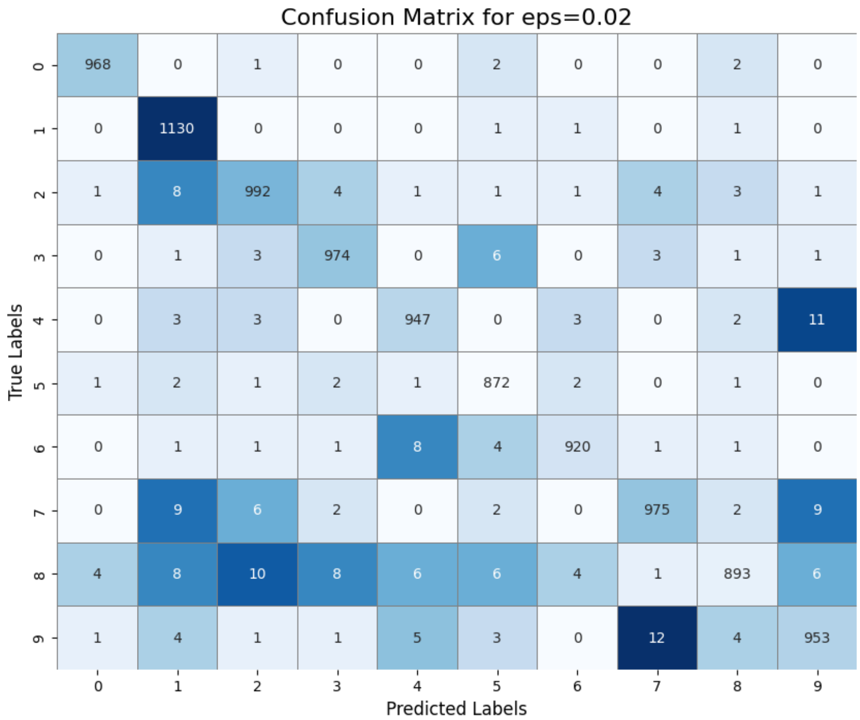
```
# Apply an FGSM attack
attack = fb.attacks.LinfPGD()
epsilons = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
raw_advs, clipped_advs, success = attack(fmodel, images, labels, epsilons=epsilons)
```

```
# Assuming 'clipped_advs' are the adversarial examples for different epsilons
for eps, advs_ in zip(epsilons, clipped_advs):
    # Predict the labels of the adversarial examples
    y_adv = np.argmax(model.predict(advs_), axis=1)

    # Calculate accuracy on adversarial examples
    accuracy_adv = np.mean(y_adv == correct_labels)
    print(f"\nAdversarial 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
    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
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey')

    # 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)

    # Save the plot
    image_filename = f'confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()  # Display the figure in the notebook
```

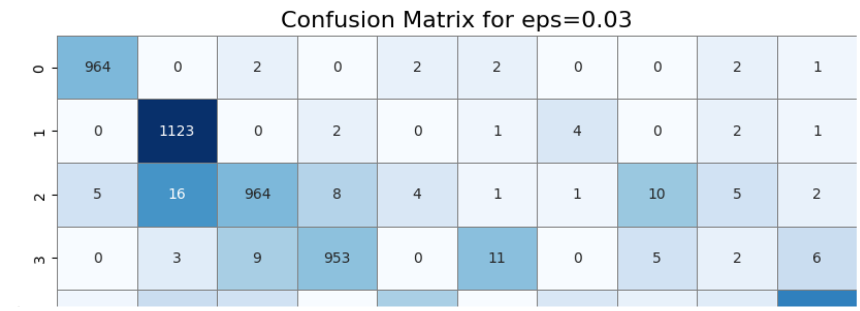
```
308/308 [==============================] - 1s 2ms/step
Accuracy on clean data: 100.00%
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.20%
```




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.86%
```

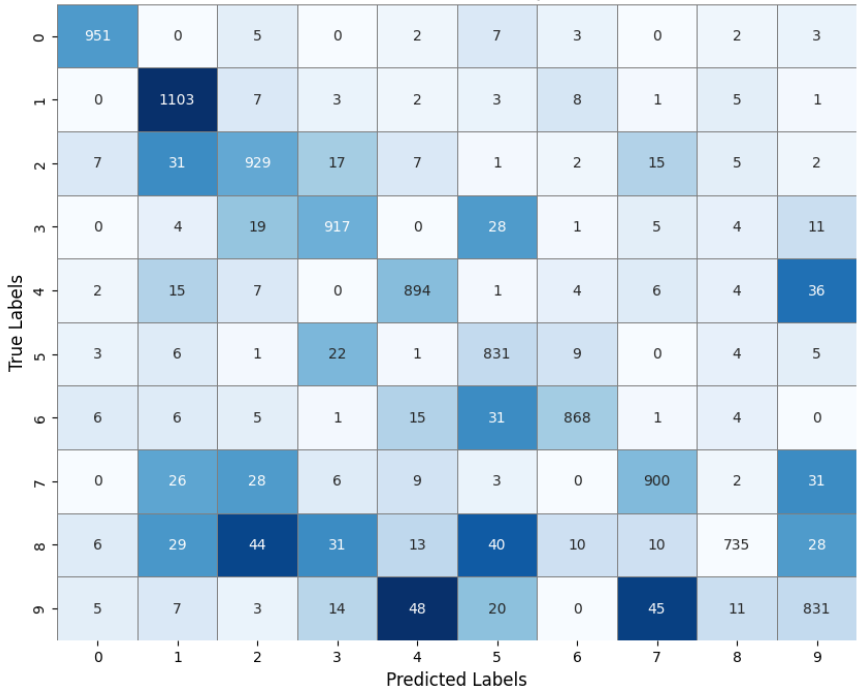



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.41%
```



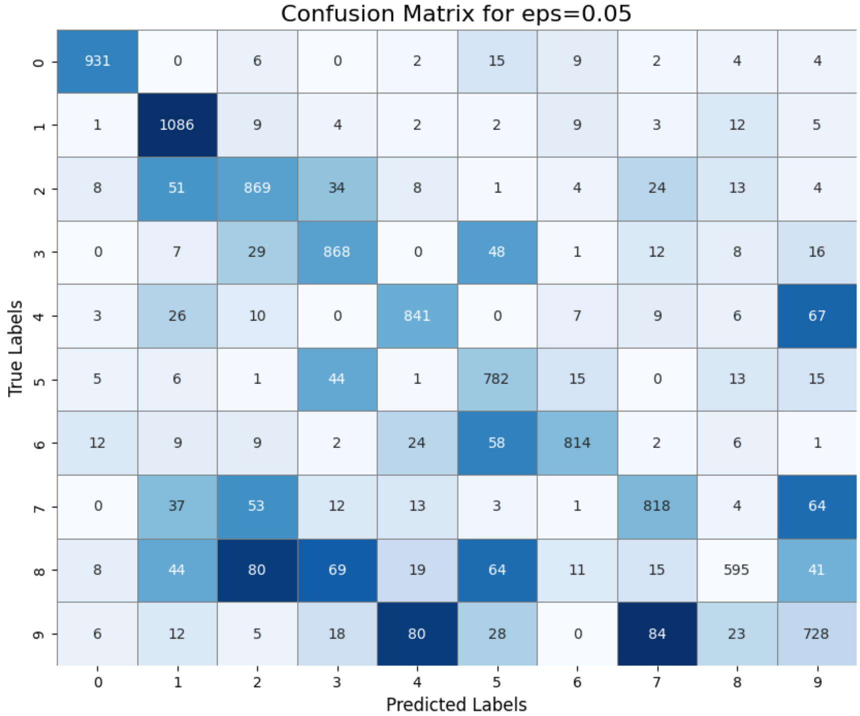

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.10%
```

Confusion Matrix for eps=0.04




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.73%
```

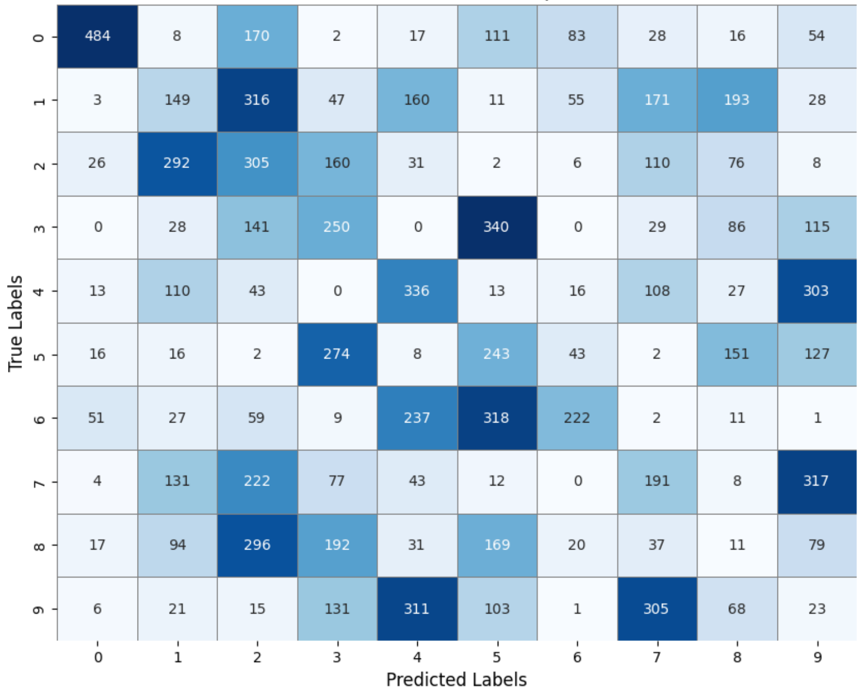
Confusion Matrix for eps=0.05




```
308/308 [==============================] - 1s 2ms/step
```

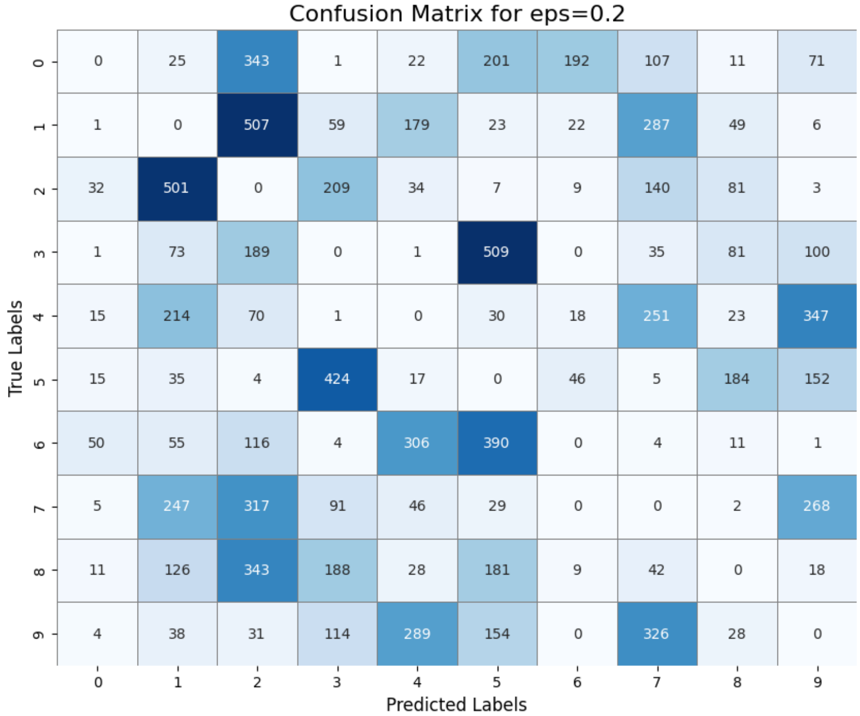
```
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.51%
```

Confusion Matrix for eps=0.1



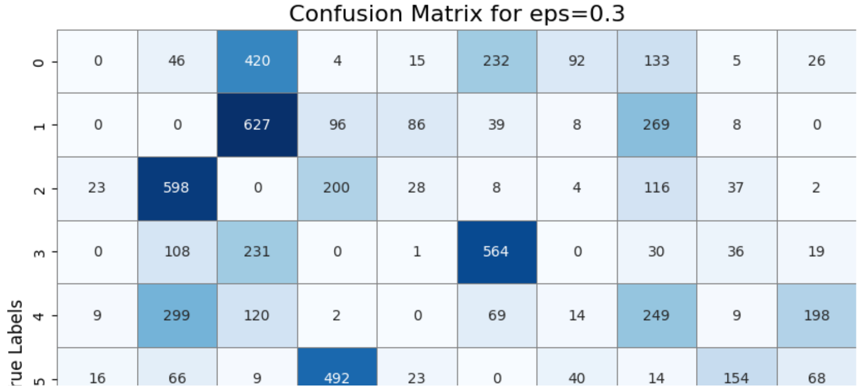

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

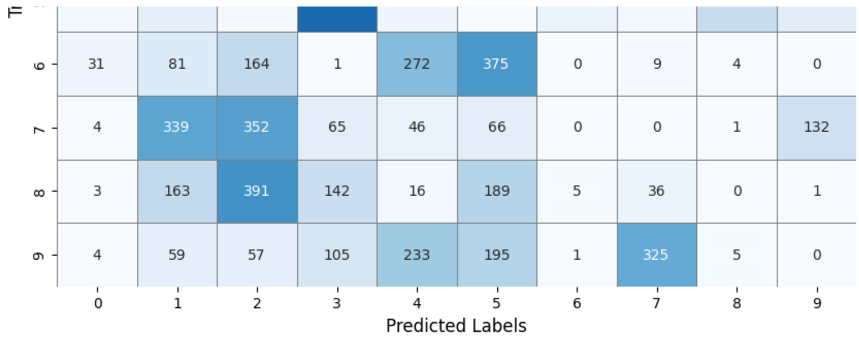
Confusion Matrix for eps=0.2



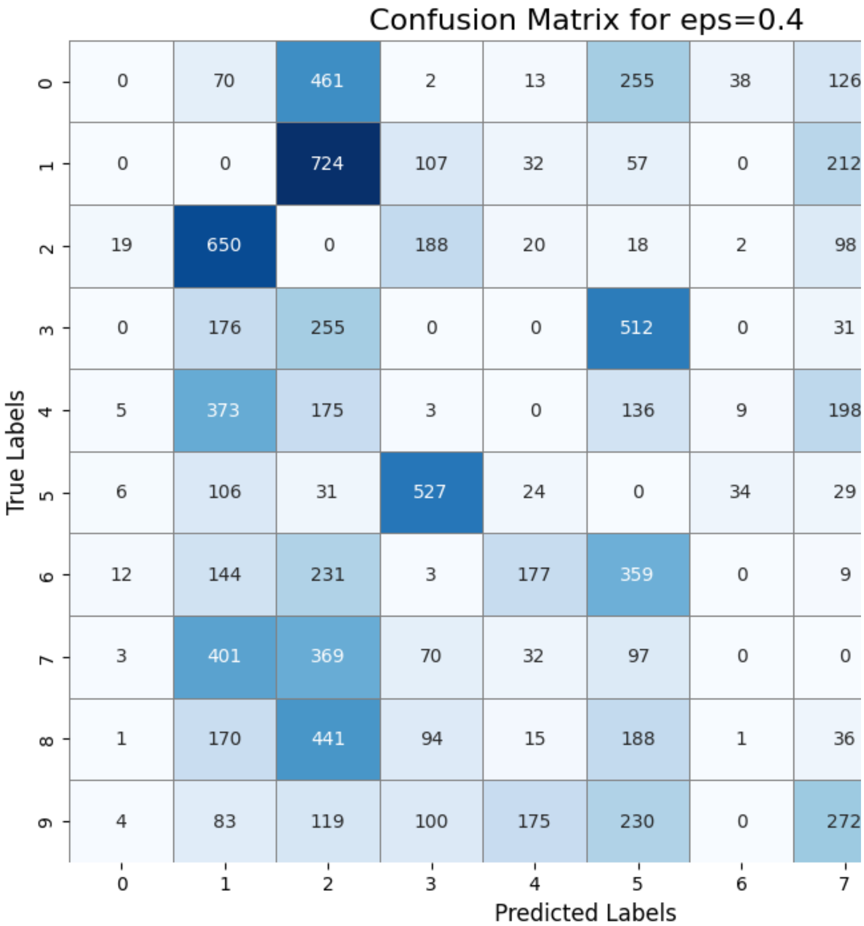

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.3

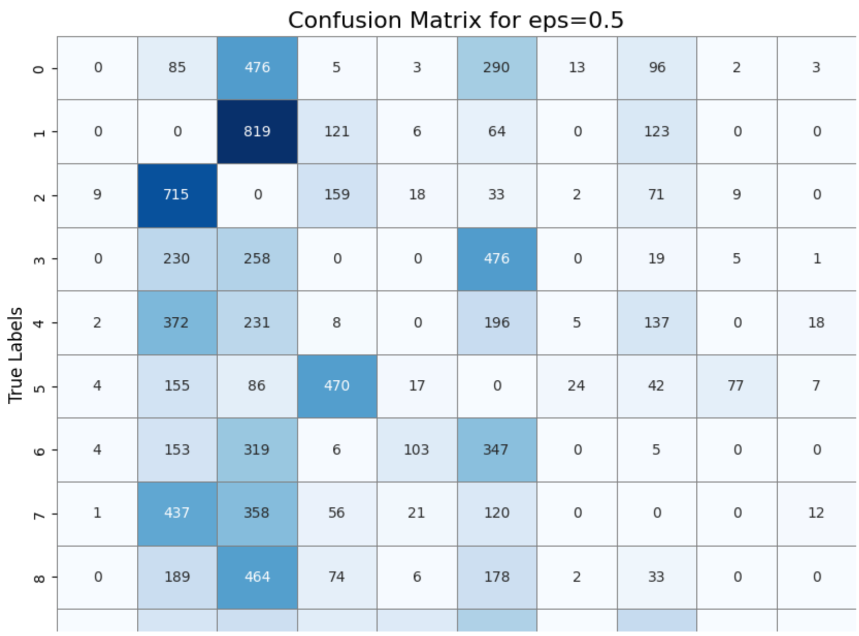


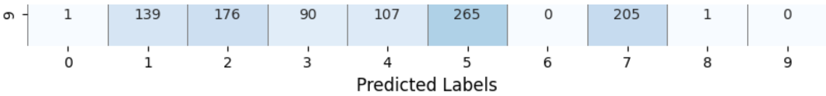



```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

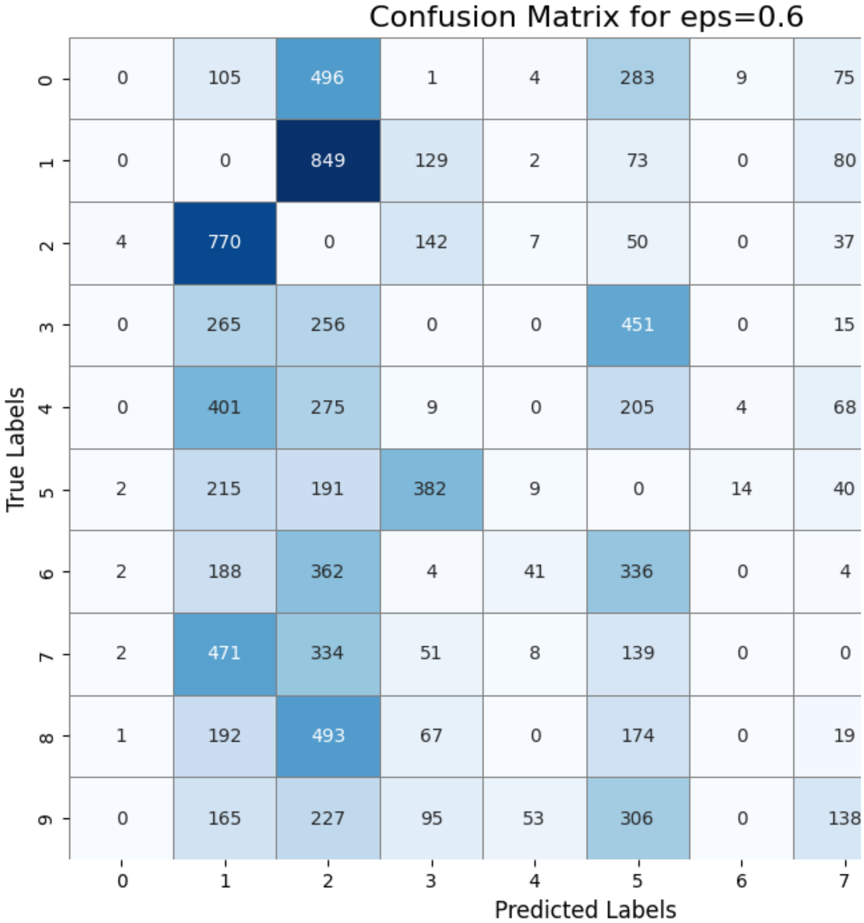



```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```






```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

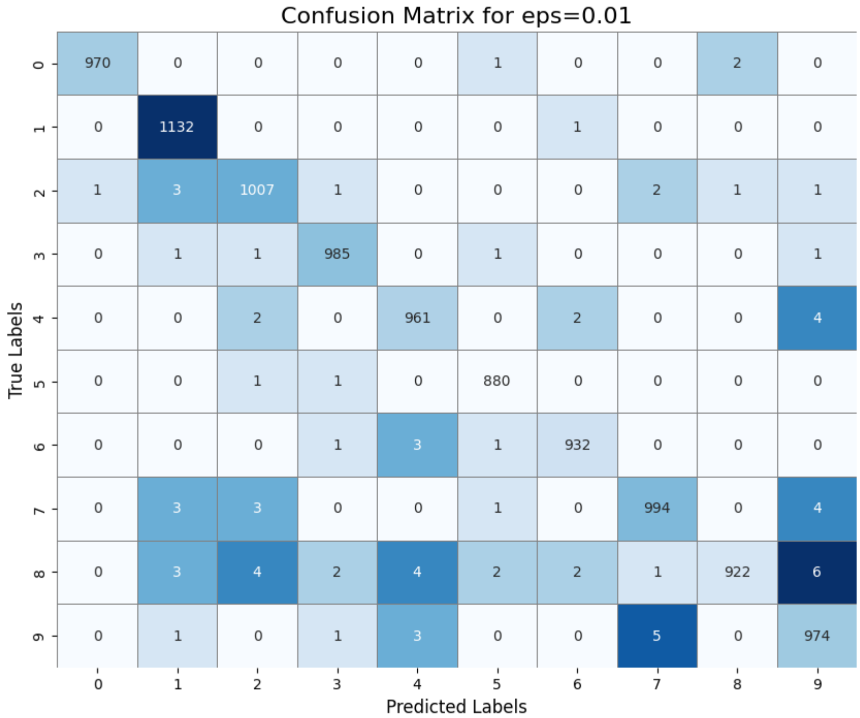


## 5 times ProjectedGradientDescentAttack

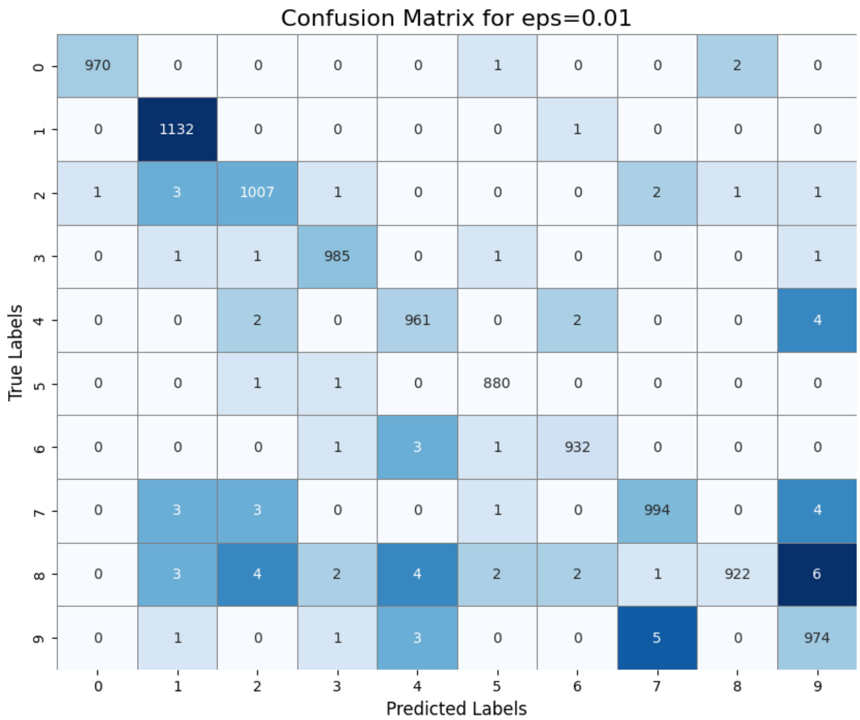
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
import foolbox as fb
import eagerpy as ep
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

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

# Load and preprocess the examples
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy').astype('int32')  # Cast labels to int32

# Create a Foolbox model
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))



# Calculate accuracy on clean data
clean_predictions = model.predict(correct_examples).argmax(axis=-1)
accuracy_clean = np.mean(clean_predictions == correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Convert data to TensorFlow tensors
images = tf.convert_to_tensor(correct_examples)
labels = tf.convert_to_tensor(correct_labels, dtype=tf.int32)  # Cast labels to int32

# Create a Foolbox model for TensorFlow
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

# Apply an FGSM attack
attack = fb.attacks.LinfPGD()
epsilons = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
raw_advs, clipped_advs, success = attack(fmodel, images, labels, epsilons=epsilons)




# Assuming 'clipped_advs' are the adversarial examples for different epsilons

for eps, advs_ in zip(epsilons, clipped_advs):
  for attack_num in range(1, 6):
    # Predict the labels of the adversarial examples
    y_adv = np.argmax(model.predict(advs_), axis=1)

    # Calculate accuracy on adversarial examples
    accuracy_adv = np.mean(y_adv == correct_labels)
    print(f"\nAdversarial 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
    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
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey')

    # 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
```

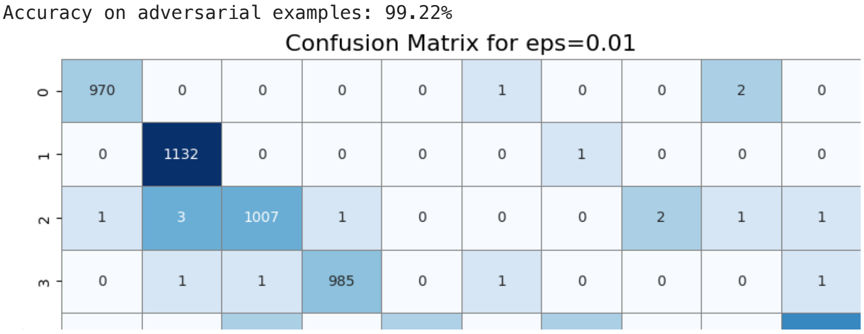
```
308/308 [==============================] - 1s 2ms/step
Accuracy on clean data: 100.00%
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.22%
```

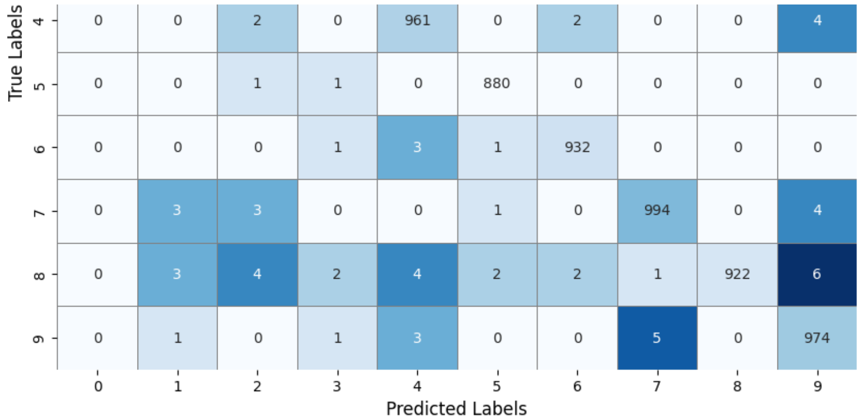



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.22%
```



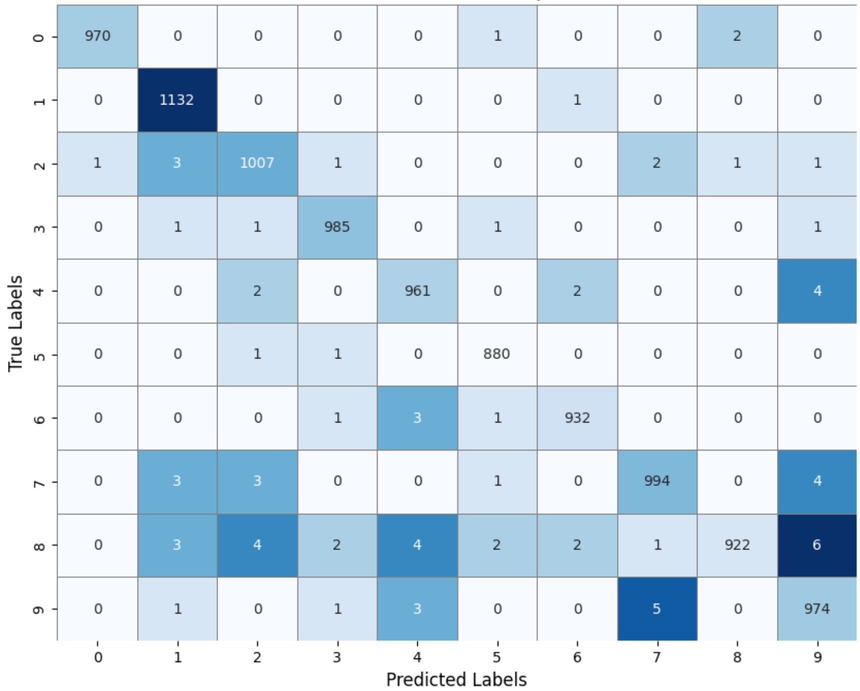

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.22%
```






```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.22%
```

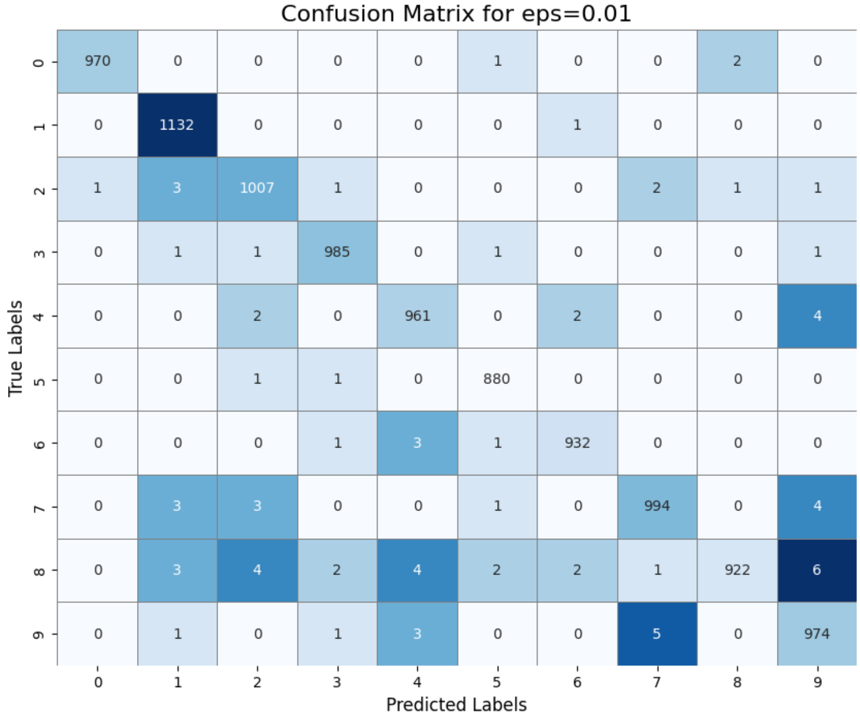
Confusion Matrix for eps=0.01





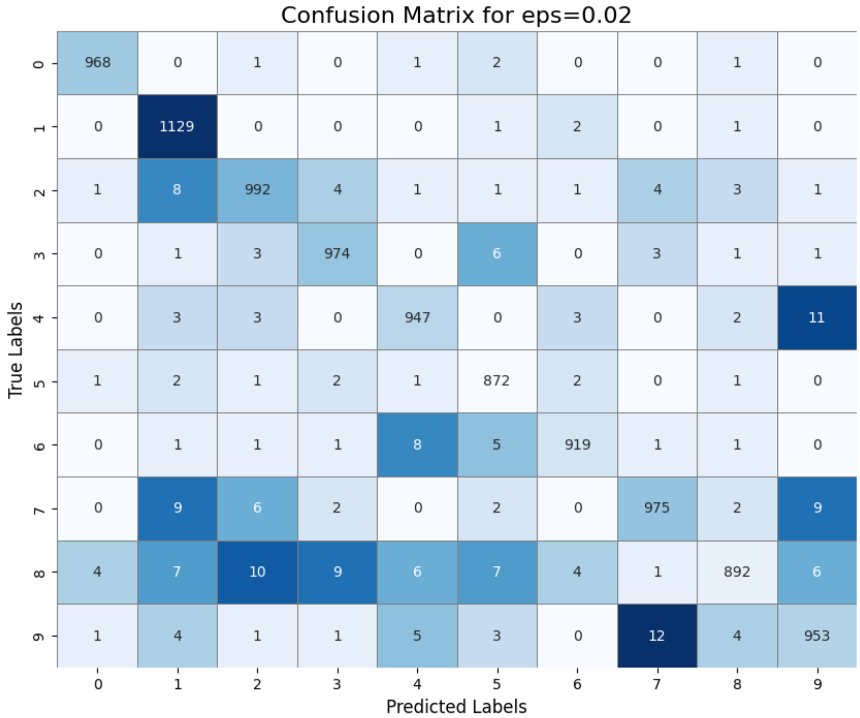
```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.22%
```

Confusion Matrix for eps=0.01



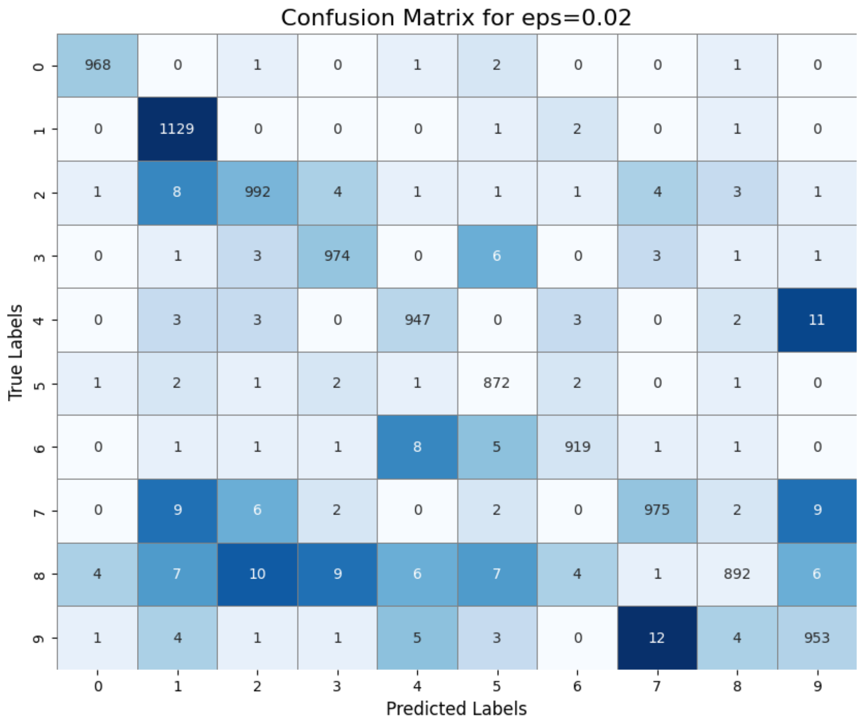

```
308/308 [==============================] - 1s 2ms/step
```

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.83%



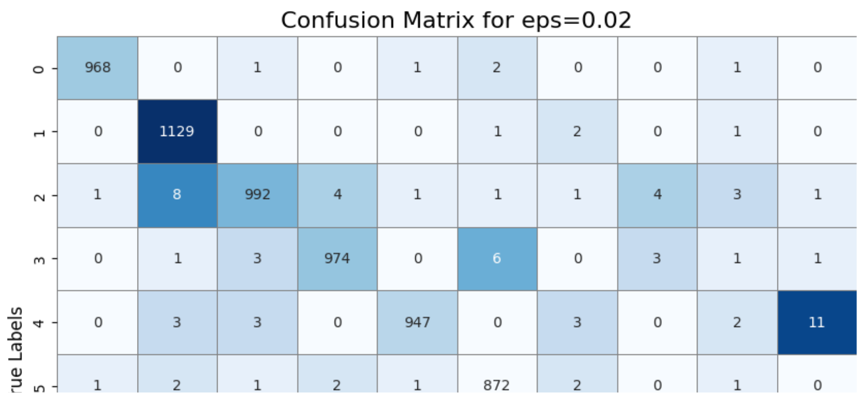

308/308 [==============================] - 1s 3ms/step

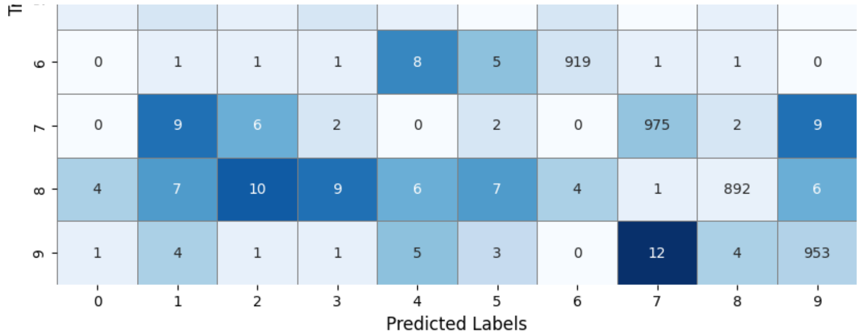
Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.83%




308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.83%

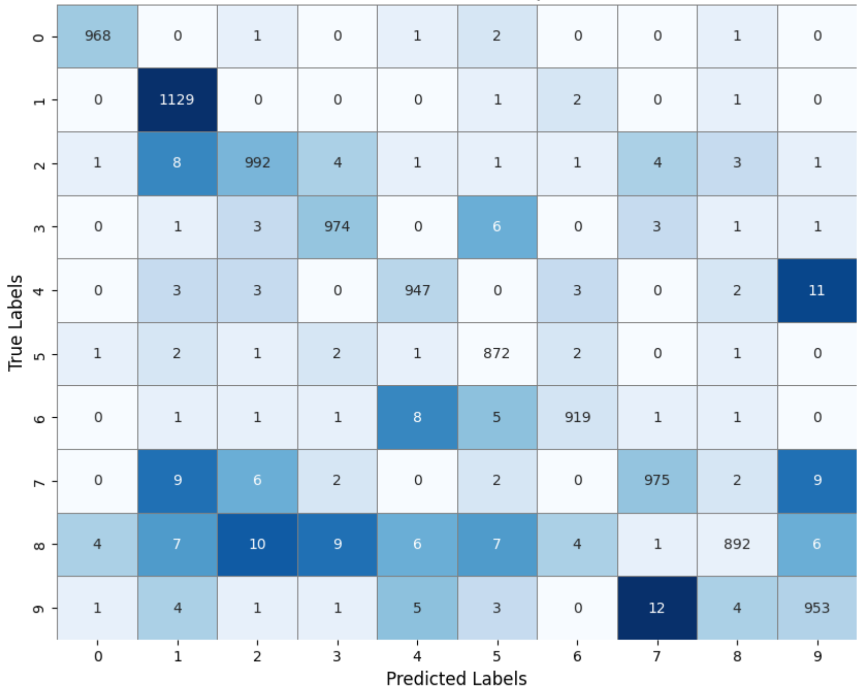





308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.83%

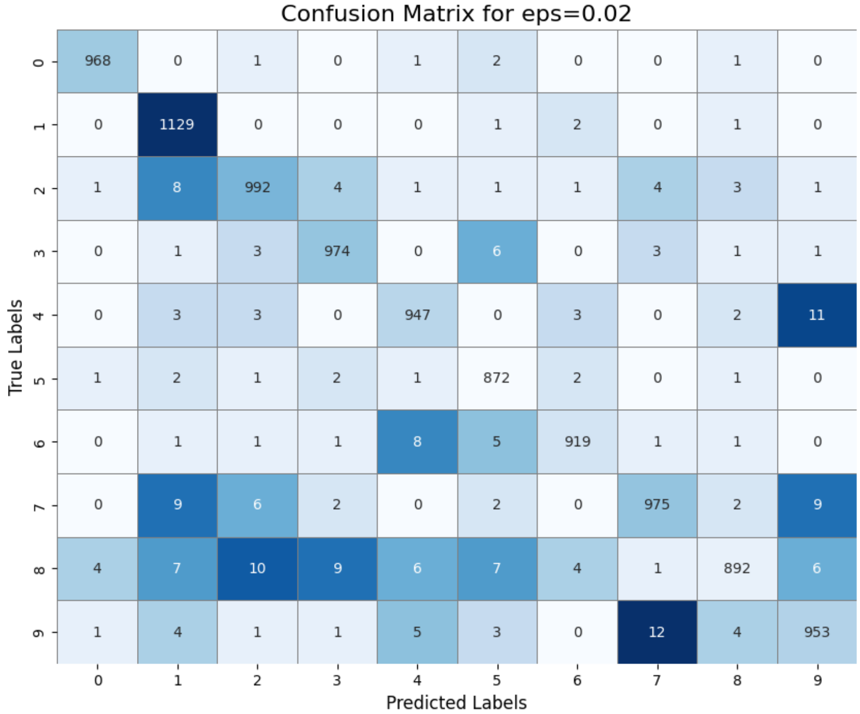
Confusion Matrix for eps=0.02




308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.83%

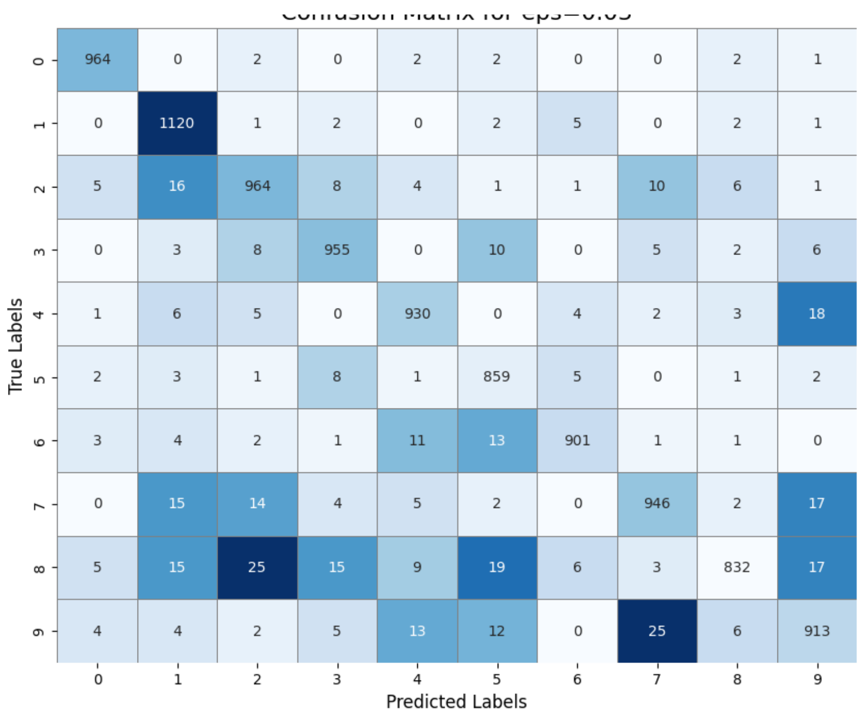
Confusion Matrix for eps=0.02




308/308 [==============================] - 1s 2ms/step

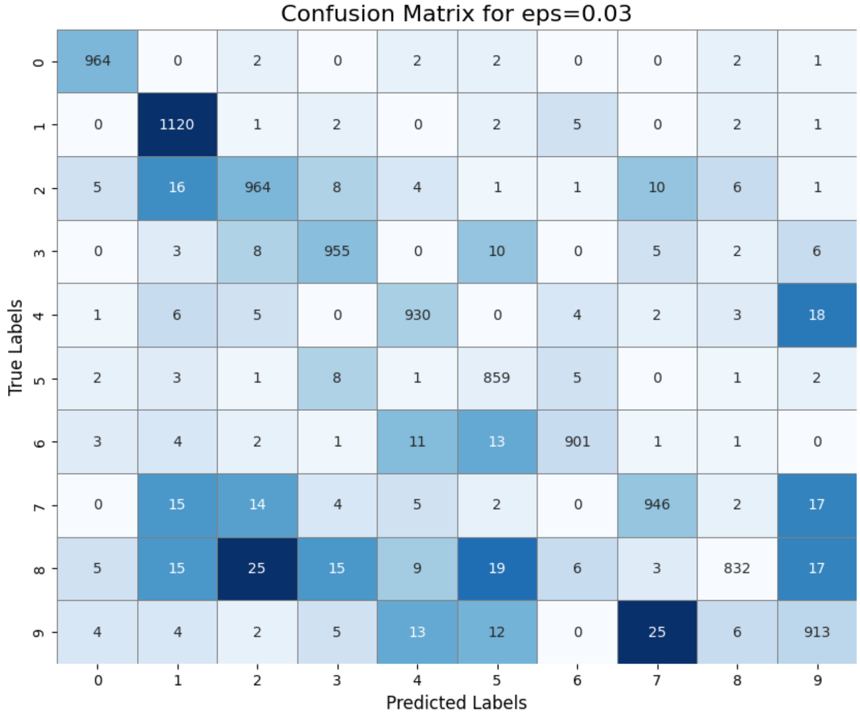
Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.42%

Confusion Matrix for eps=0.03



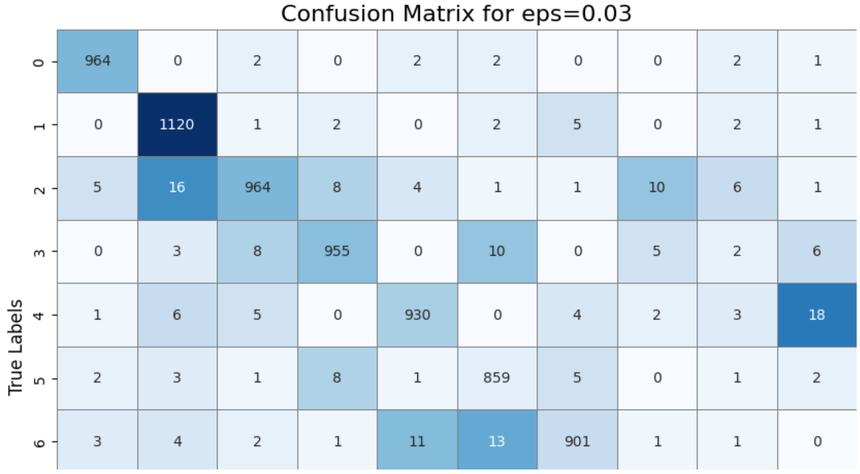

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.42%
```

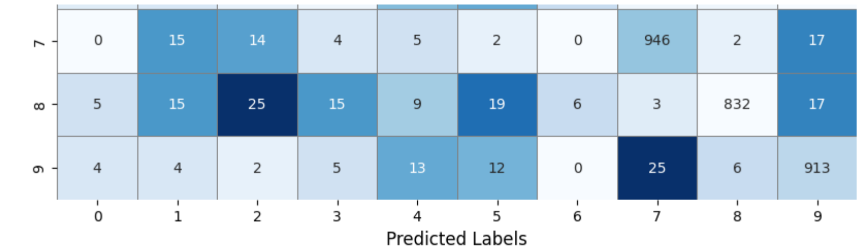
Confusion Matrix for eps=0.03




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.42%
```

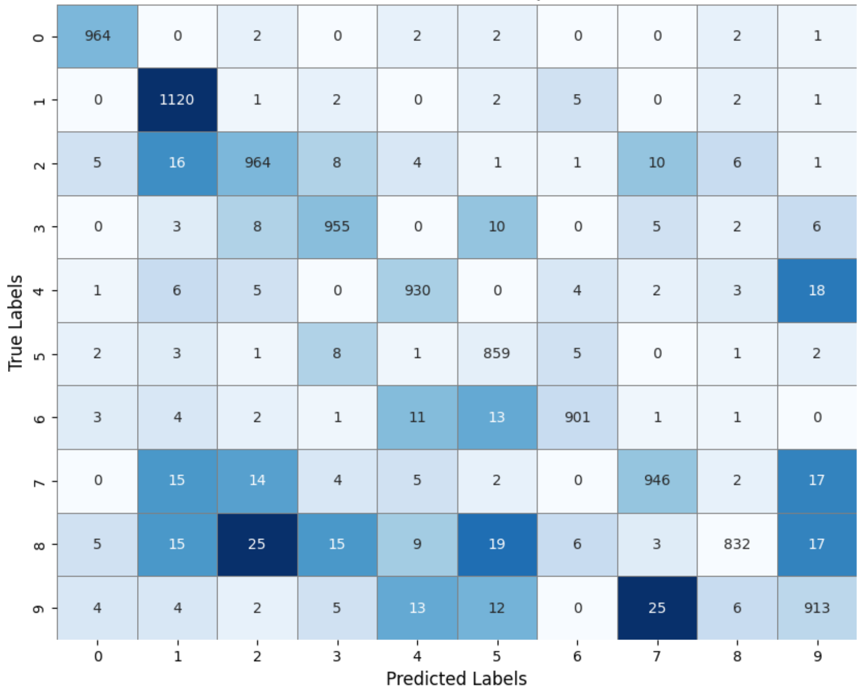
Confusion Matrix for eps=0.03





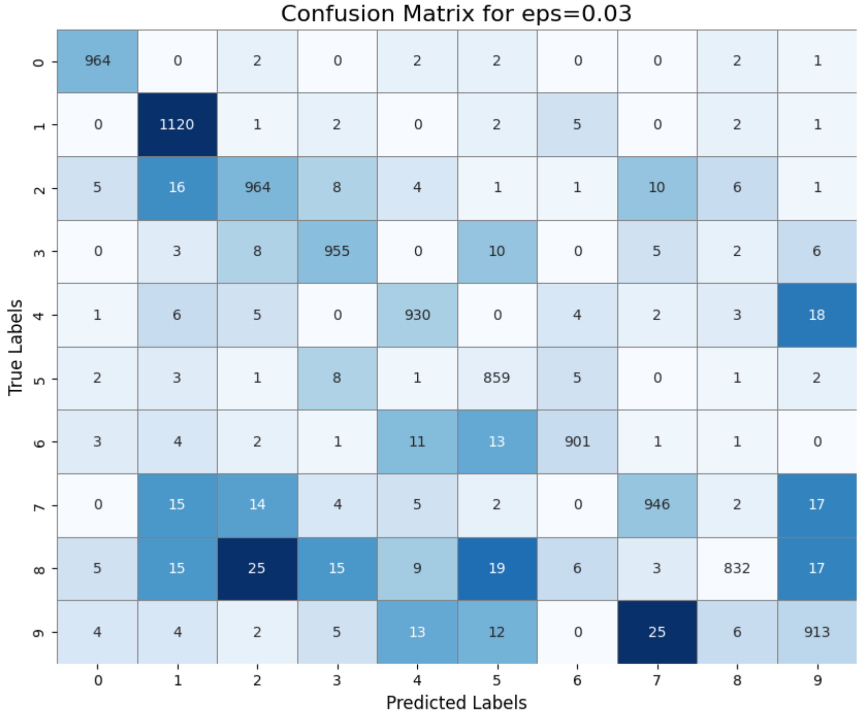

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.42%
```

Confusion Matrix for eps=0.03



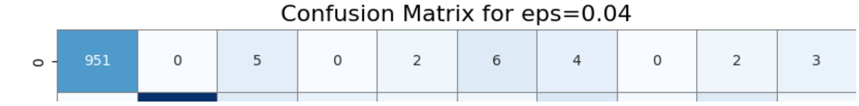

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.42%
```

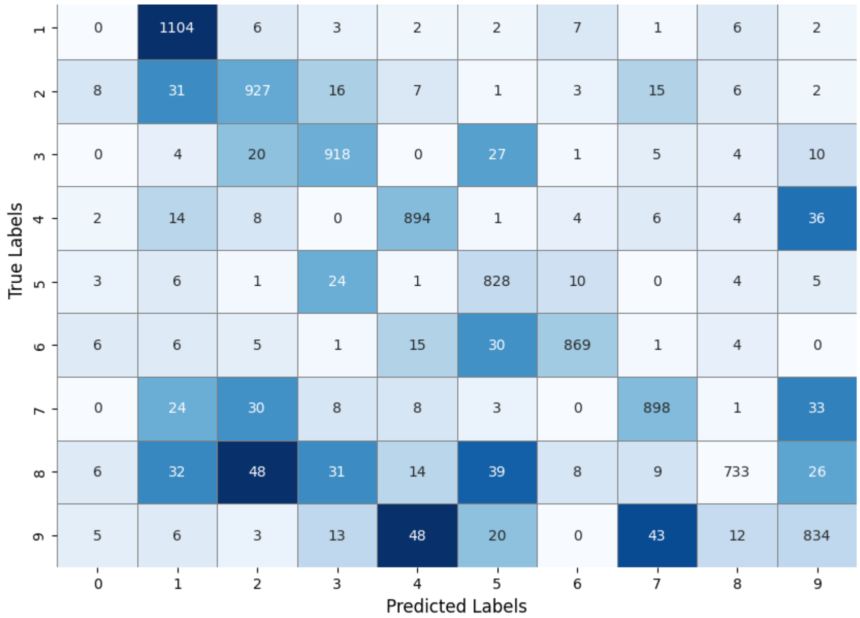
Confusion Matrix for eps=0.03




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.07%
```

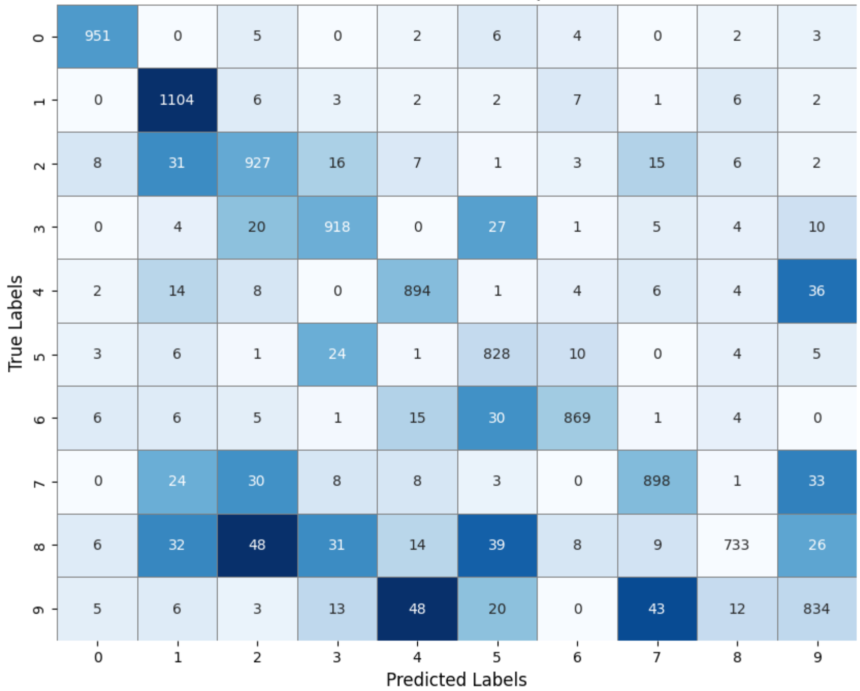
Confusion Matrix for eps=0.04





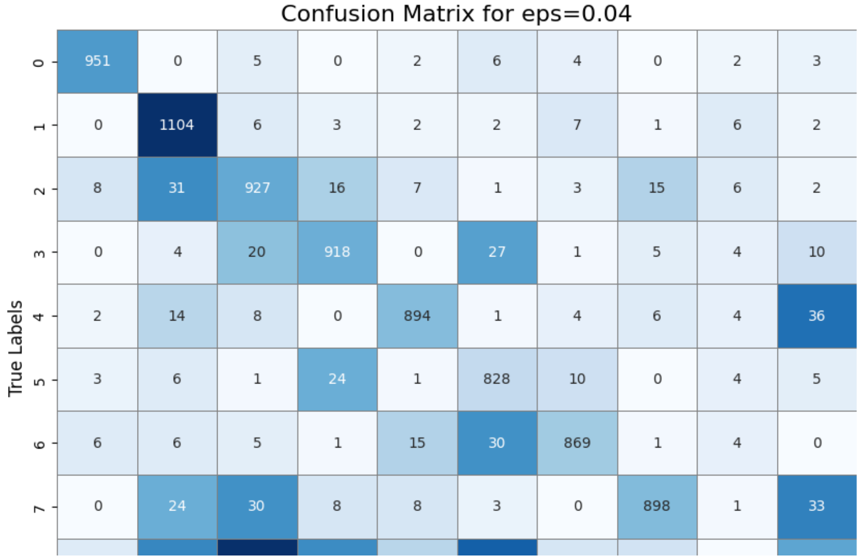

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.07%
```

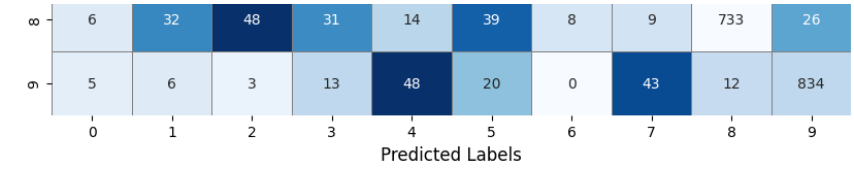
Confusion Matrix for eps=0.04




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.07%
```

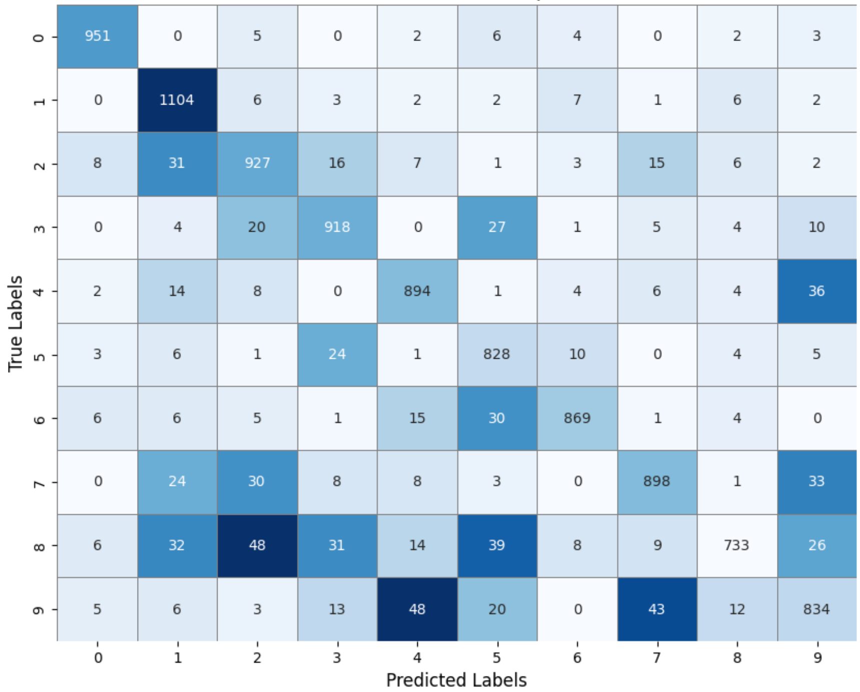
Confusion Matrix for eps=0.04





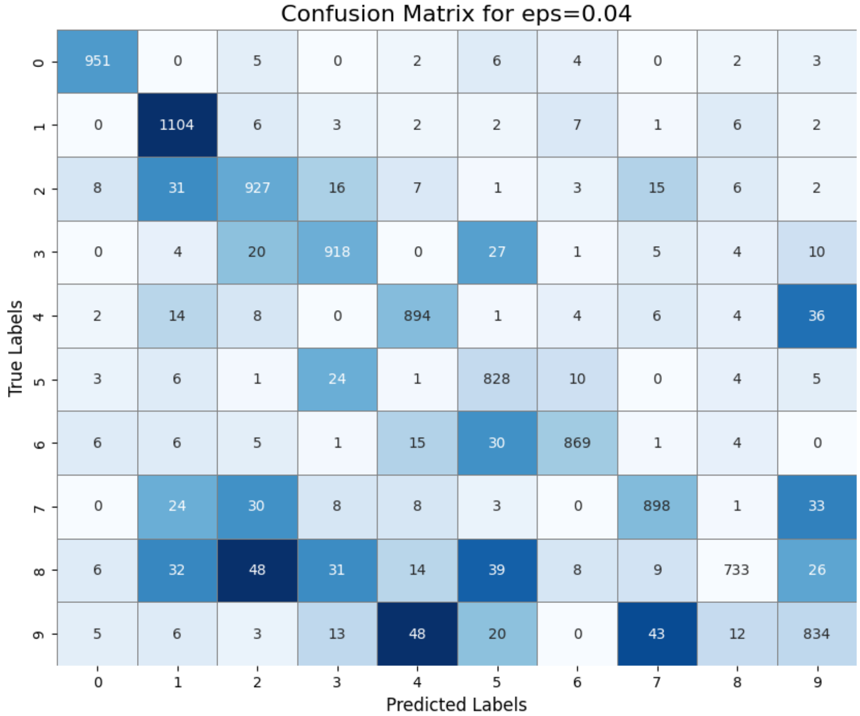

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.07%
```

Confusion Matrix for eps=0.04



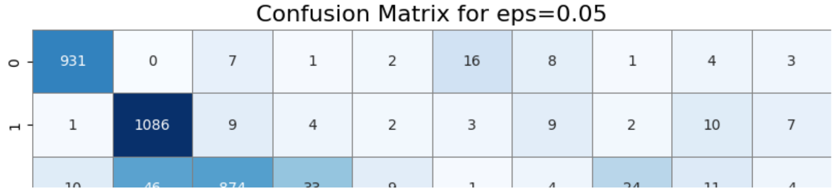

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.07%
```

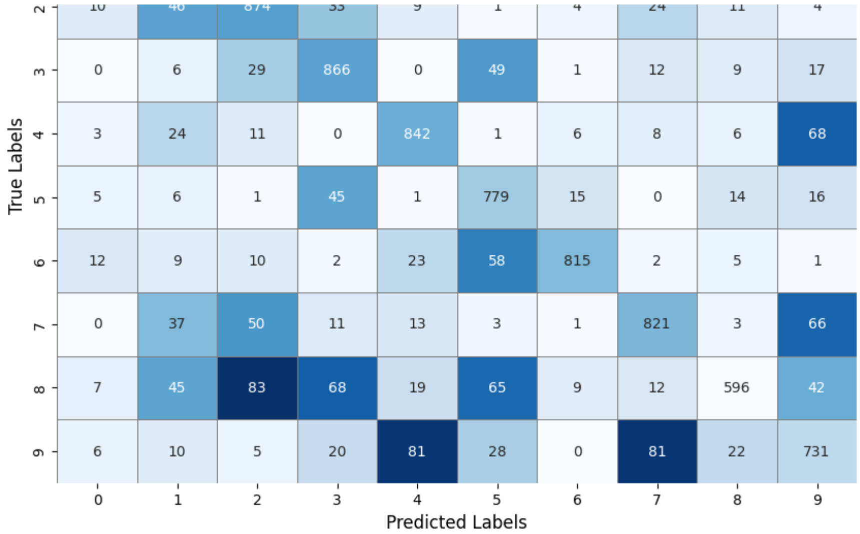
Confusion Matrix for eps=0.04




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.82%
```

Confusion Matrix for eps=0.05

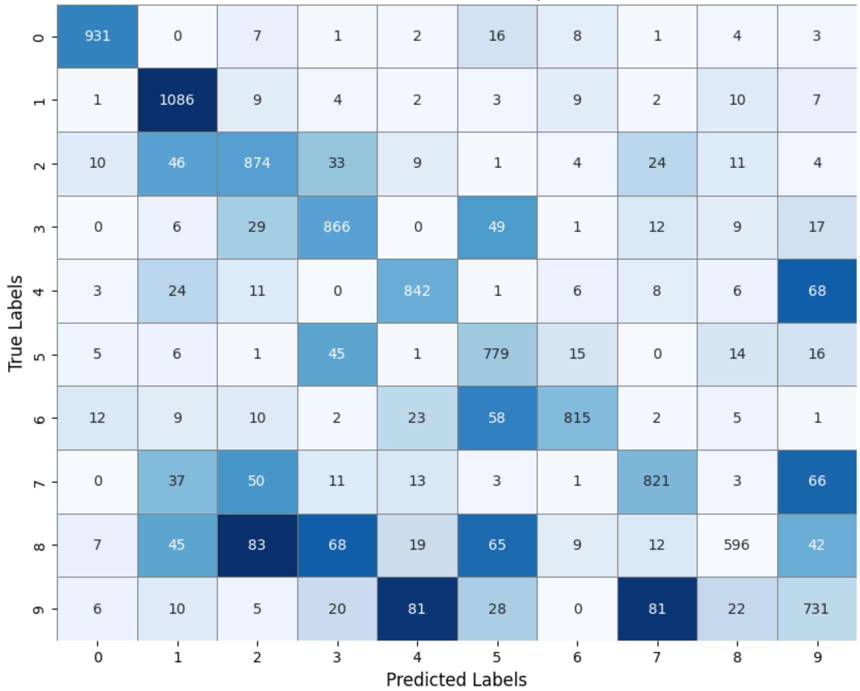





308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.82%

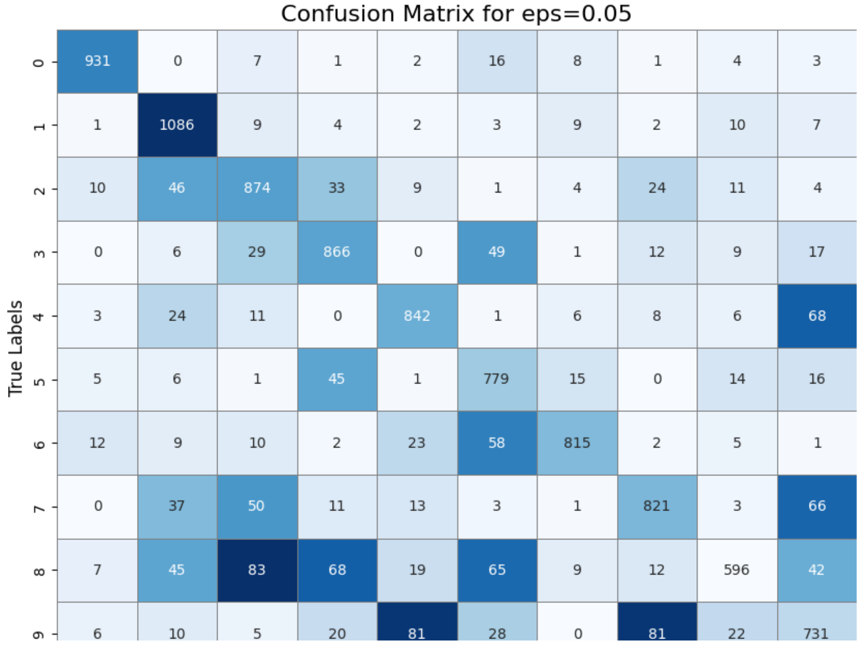
Confusion Matrix for eps=0.05

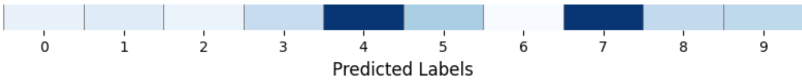



308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.82%

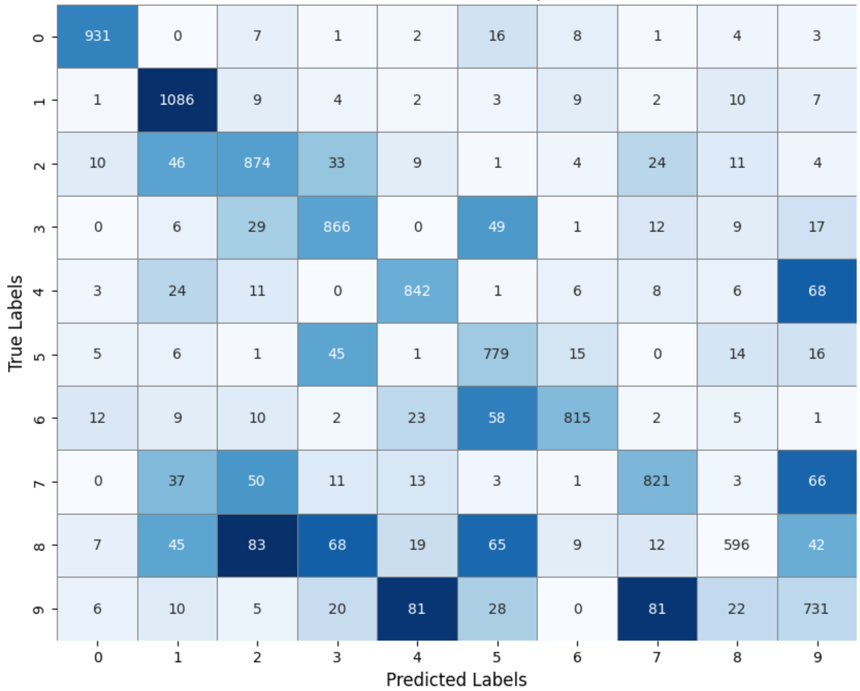
Confusion Matrix for eps=0.05






```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.82%
```

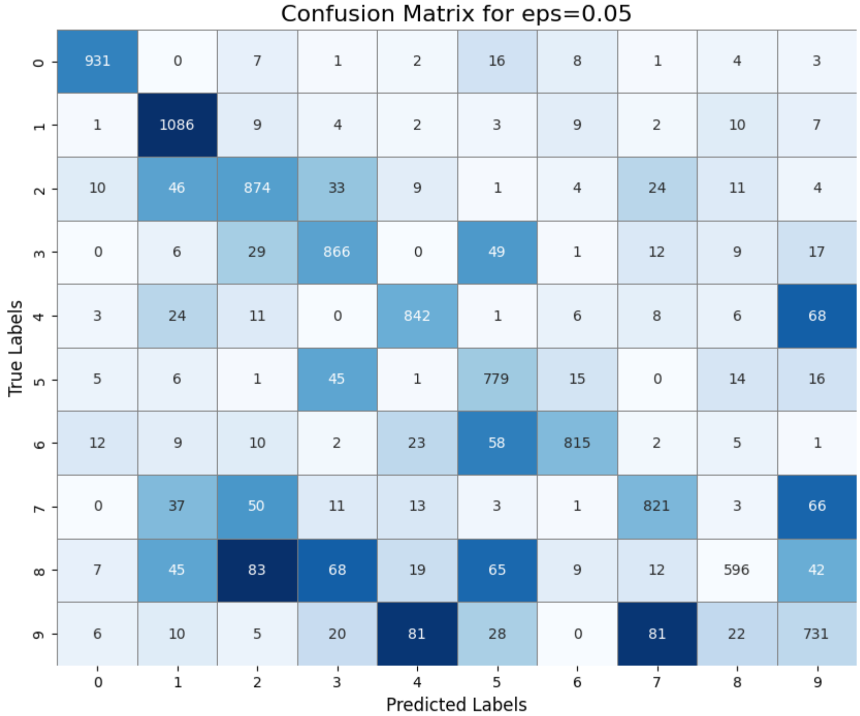
Confusion Matrix for eps=0.05




```
308/308 [==============================] - 1s 3ms/step
```

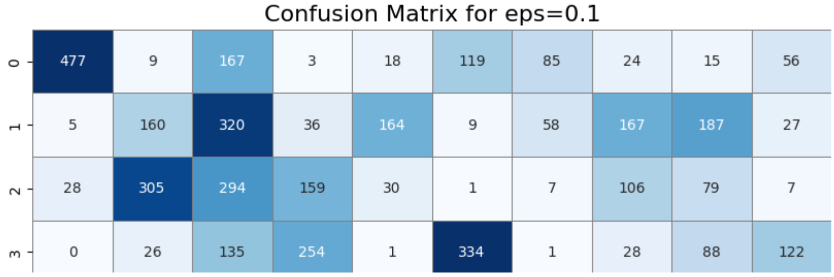

```
Adversarial test data: eps:0.05
Accuracy on adversarial examples: 84.82%
```

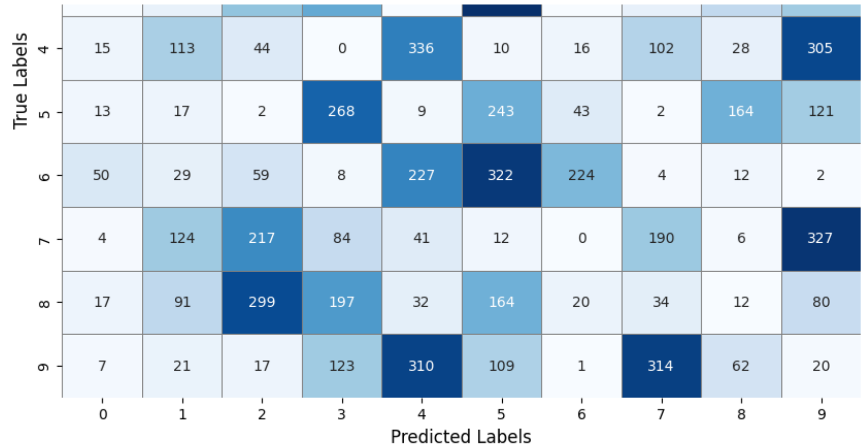
Confusion Matrix for eps=0.05

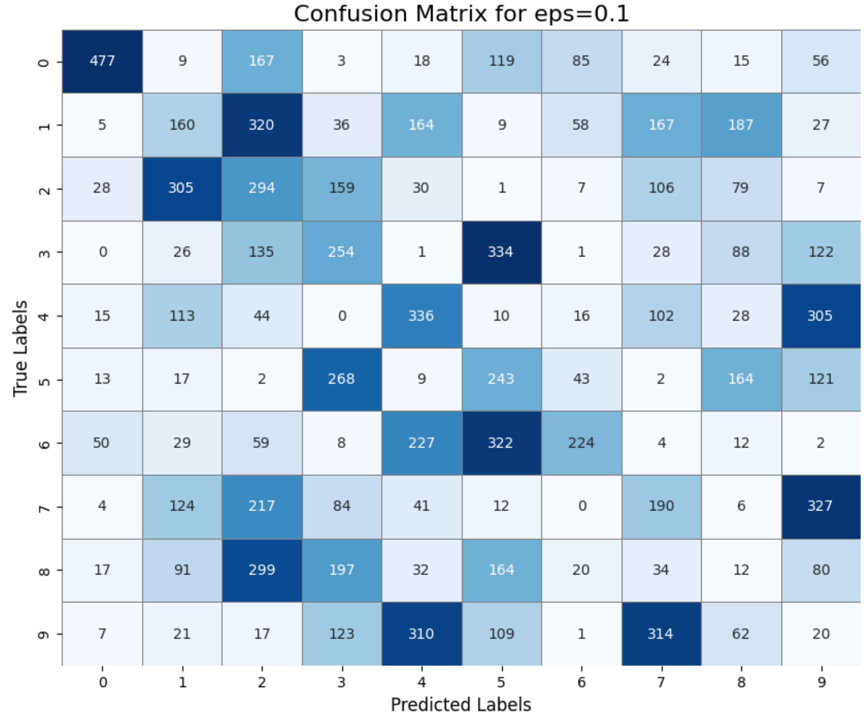

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.47%
```

Confusion Matrix for eps=0.1



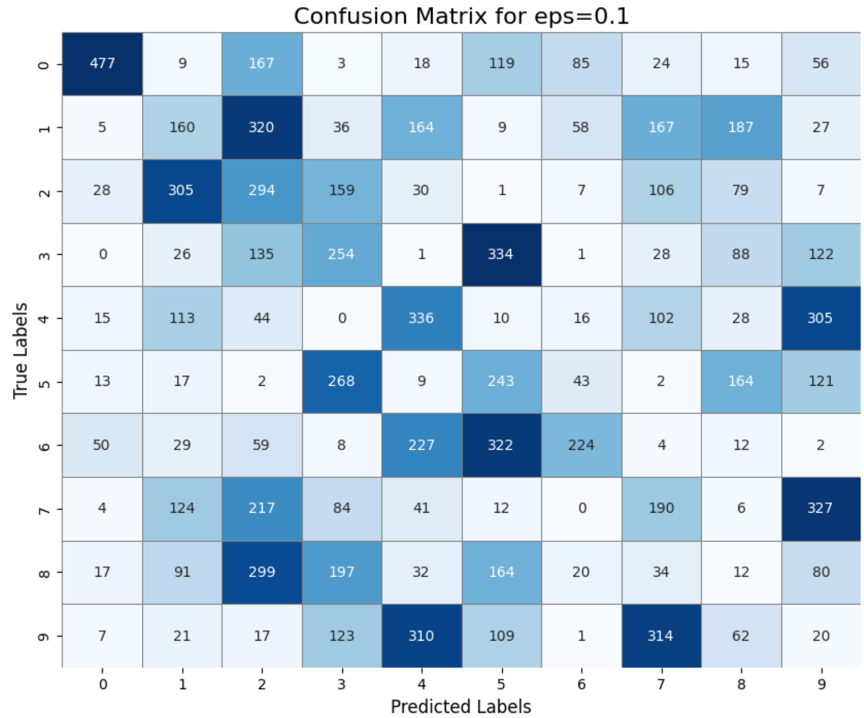



```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.47%
```



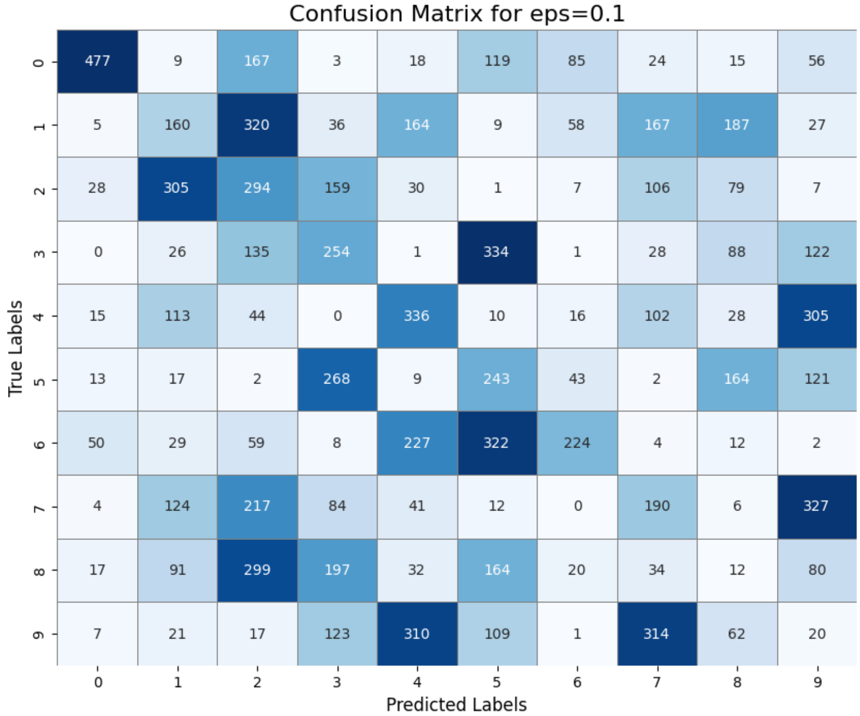


```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.47%
```

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.47%
```

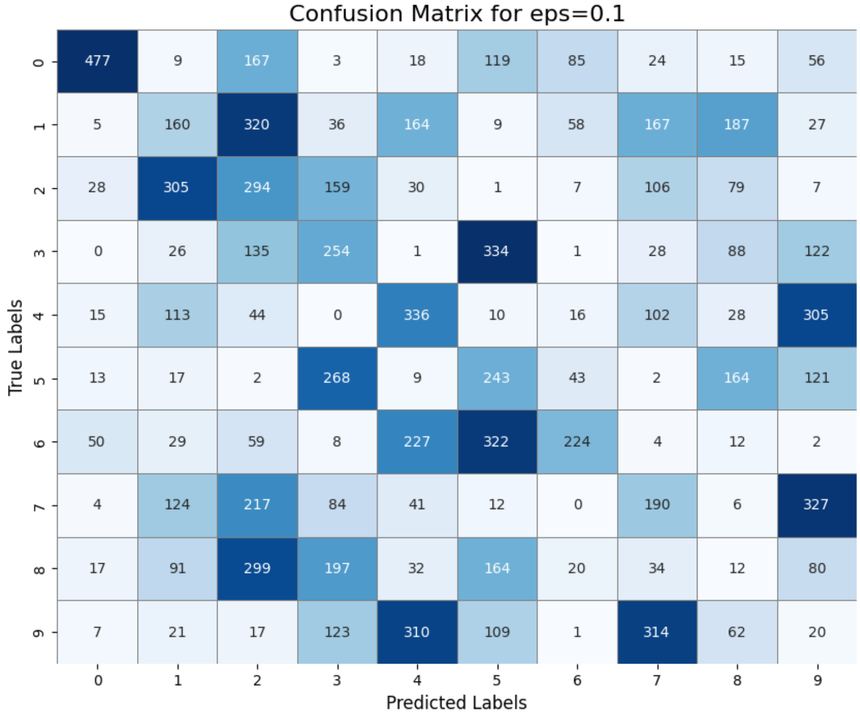
Confusion Matrix for eps=0.1





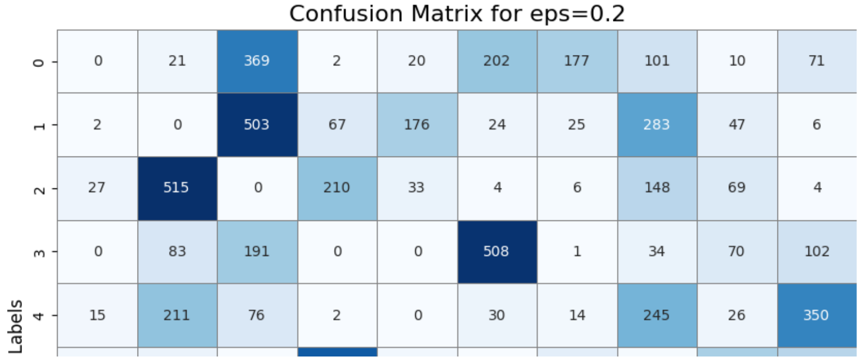
```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 22.47%
```

Confusion Matrix for eps=0.1

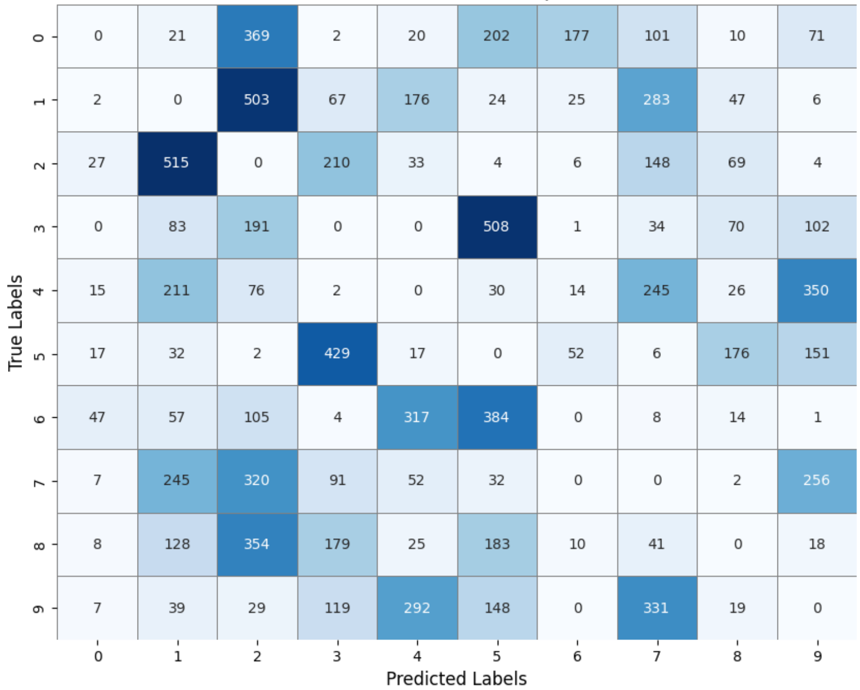

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.2



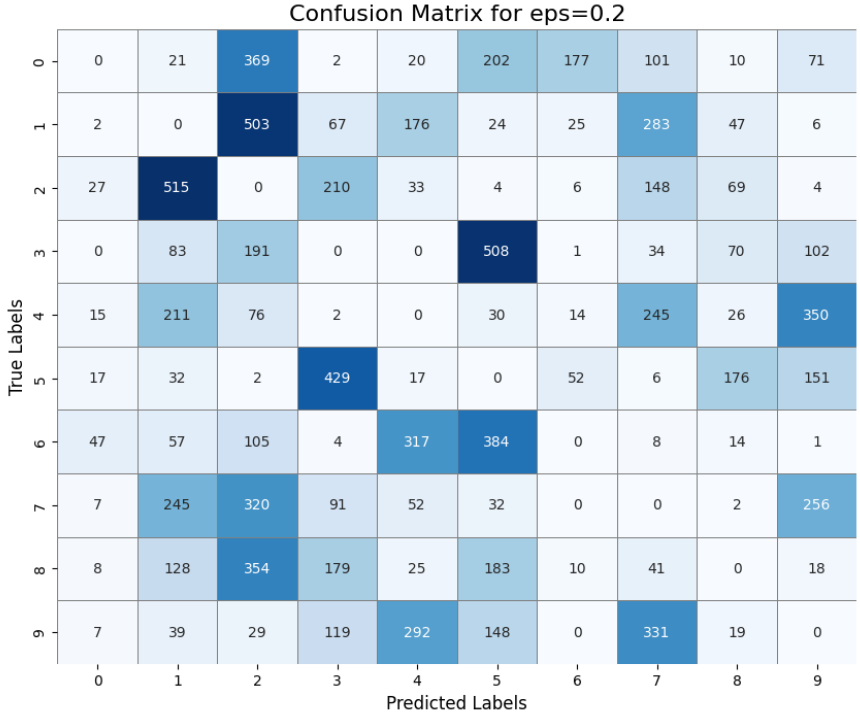

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.2



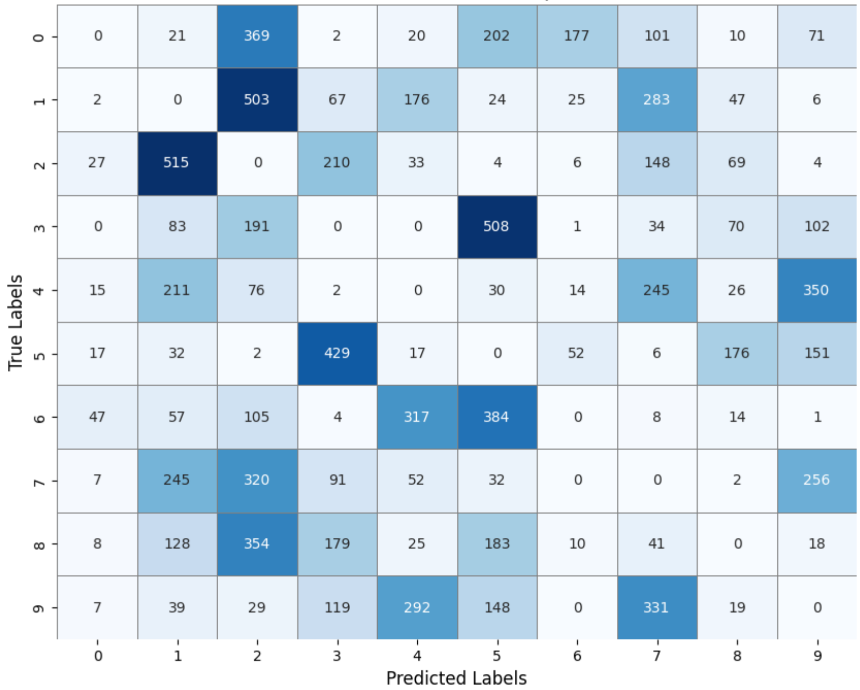

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.2



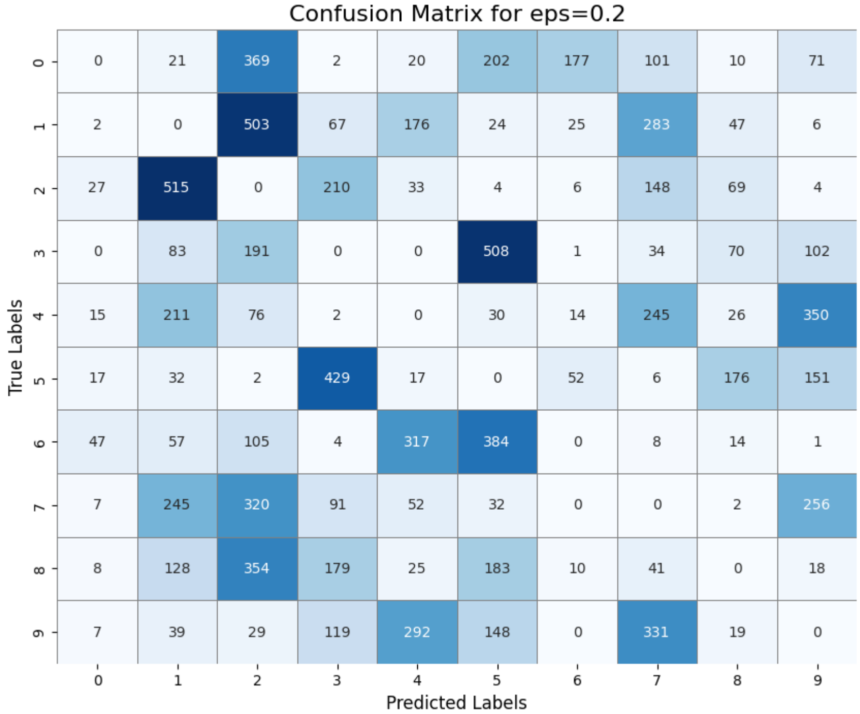

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.2



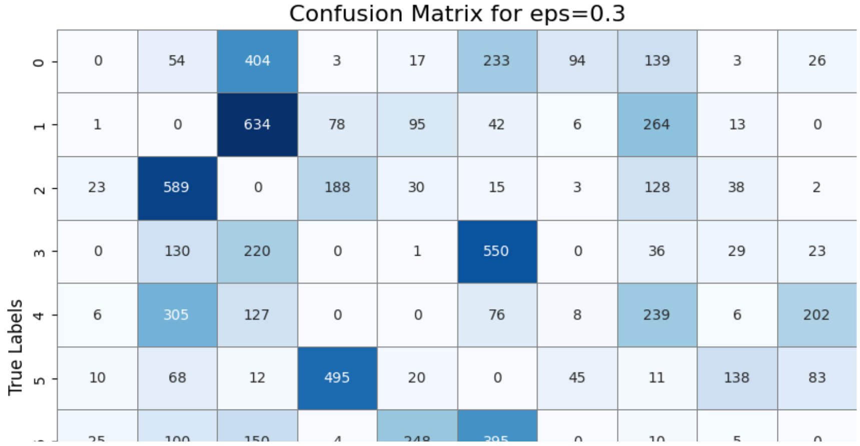

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.00%
```

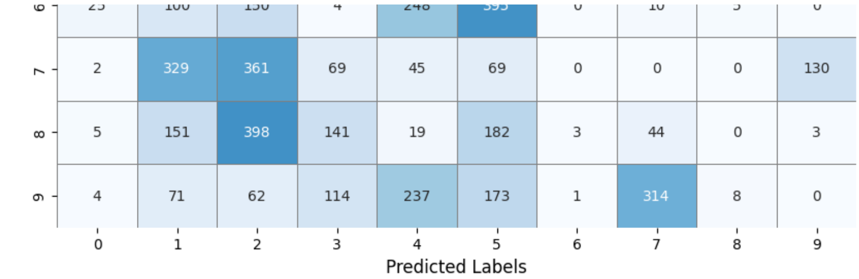
Confusion Matrix for eps=0.2




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

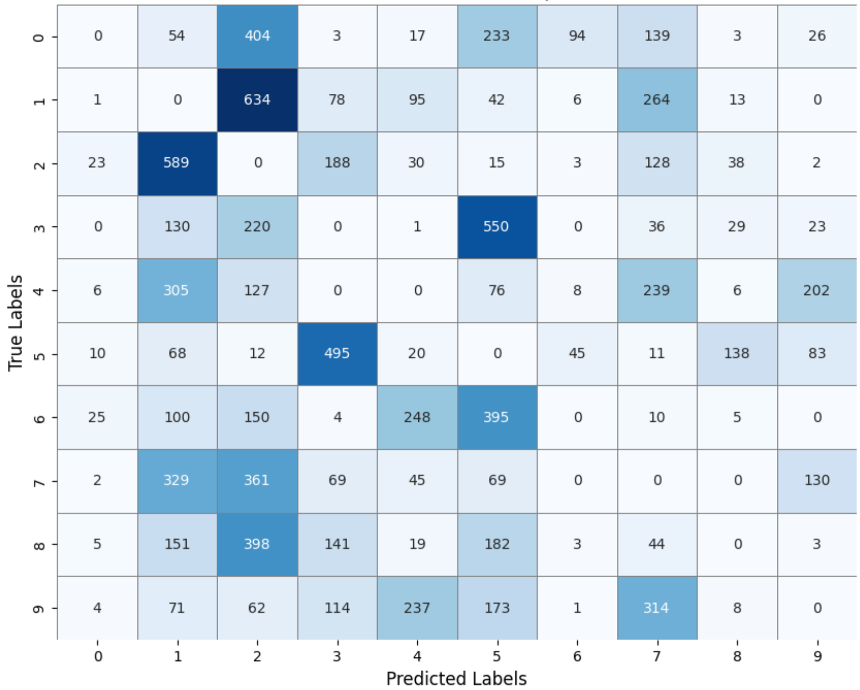
Confusion Matrix for eps=0.3





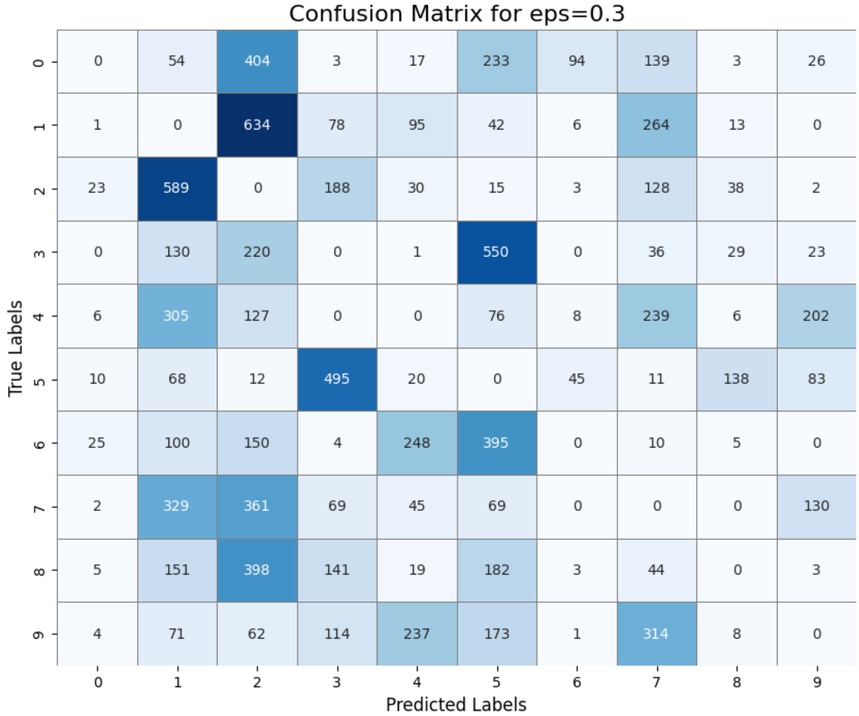

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.3



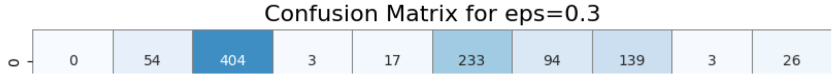

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.3



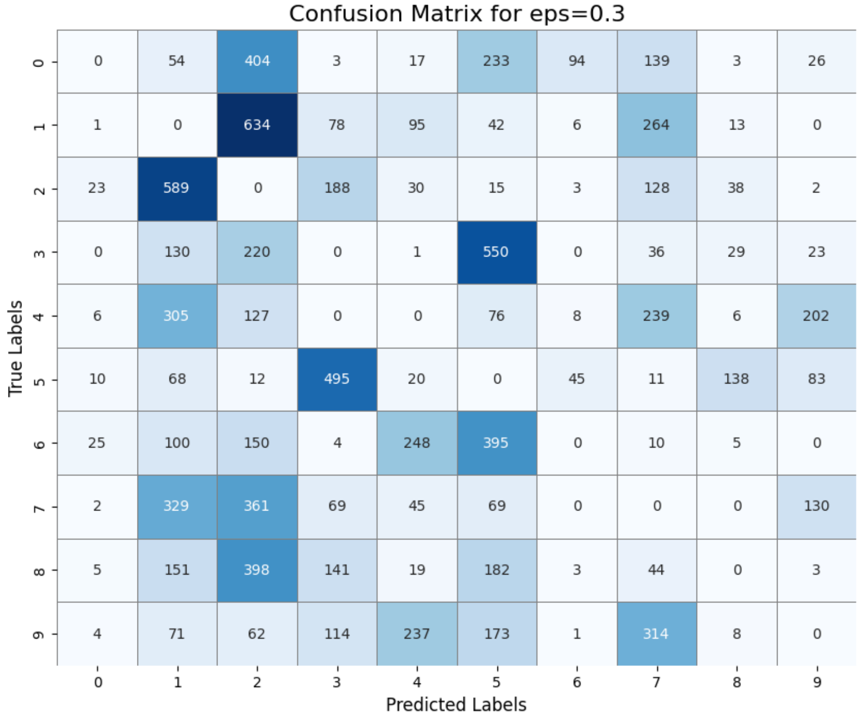

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.3



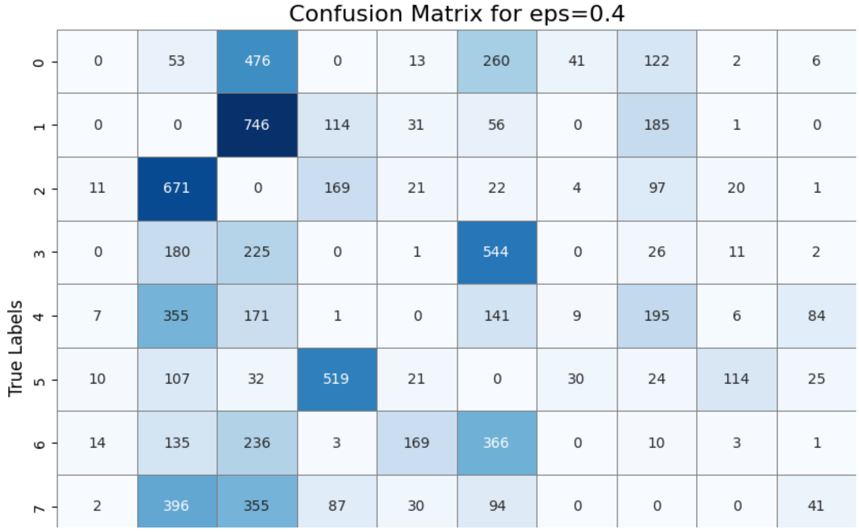

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```

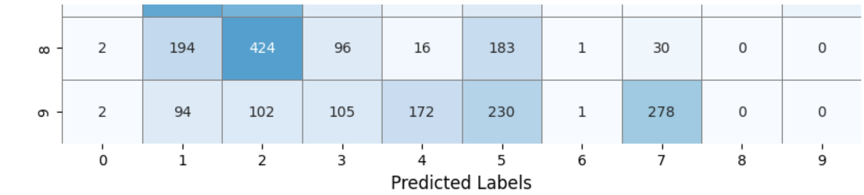
Confusion Matrix for eps=0.3




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

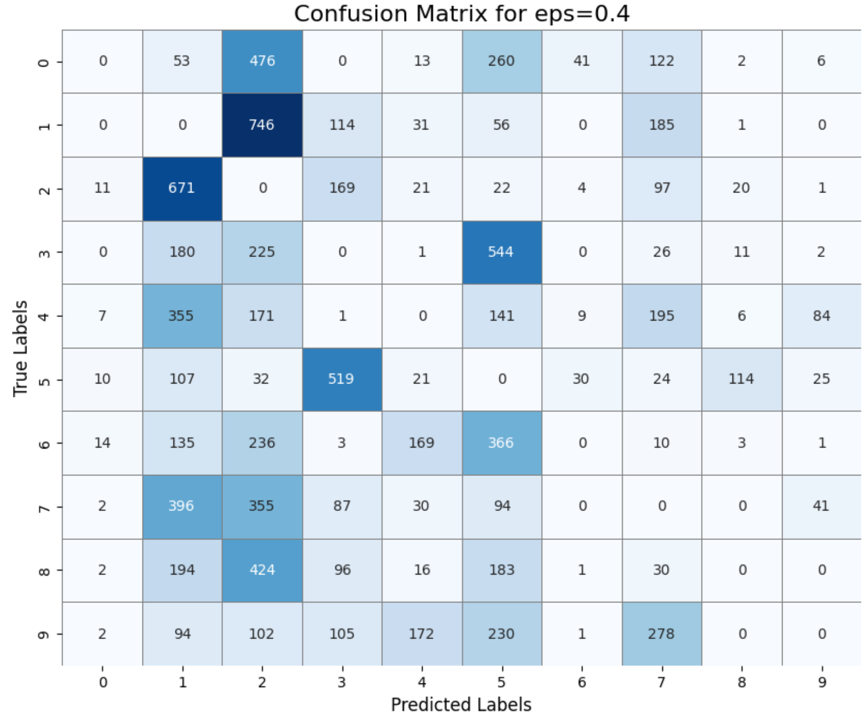
Confusion Matrix for eps=0.4





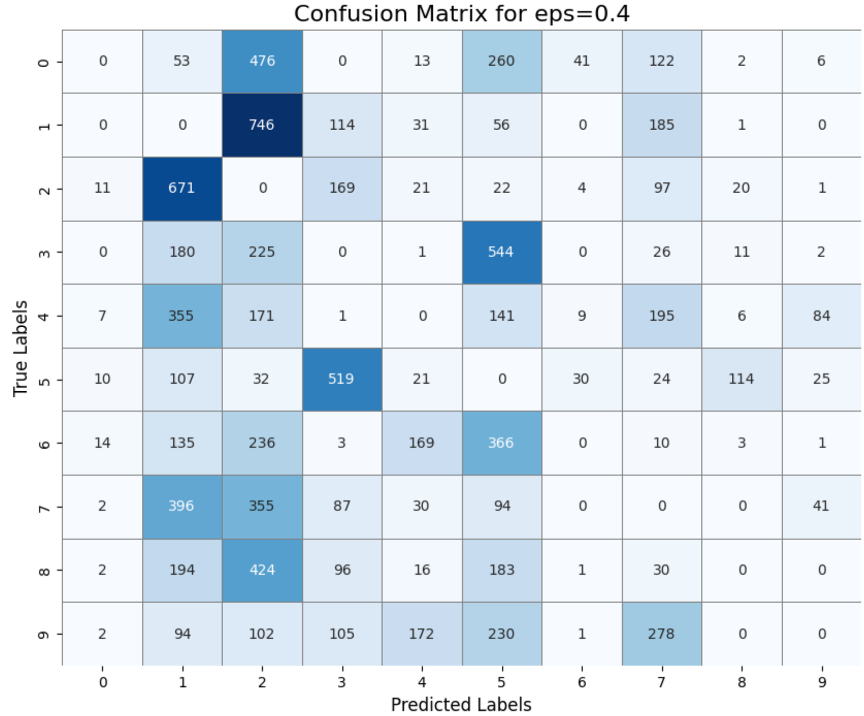

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.4



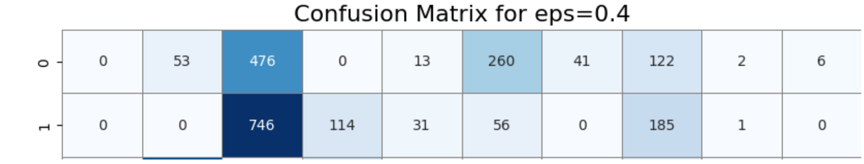

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

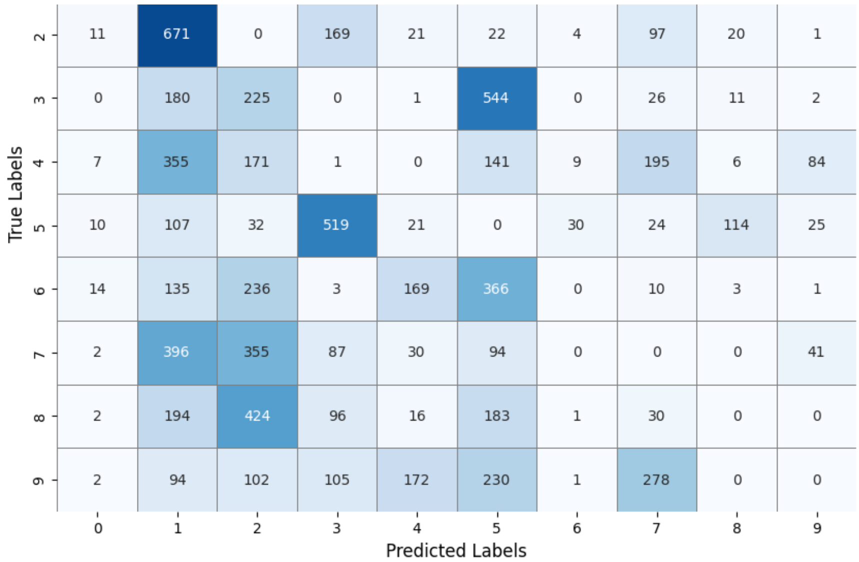
Confusion Matrix for eps=0.4




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

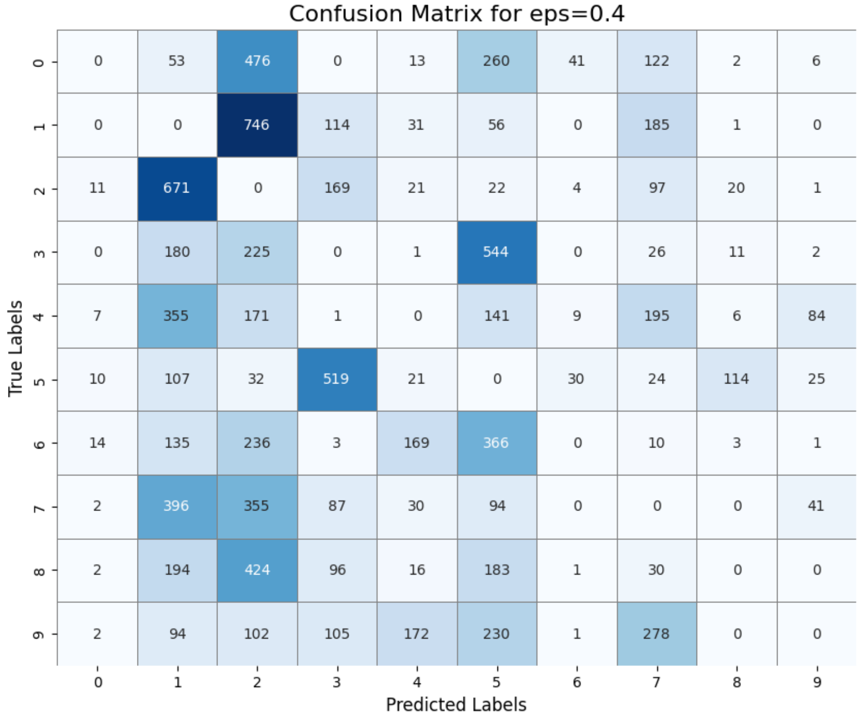
Confusion Matrix for eps=0.4





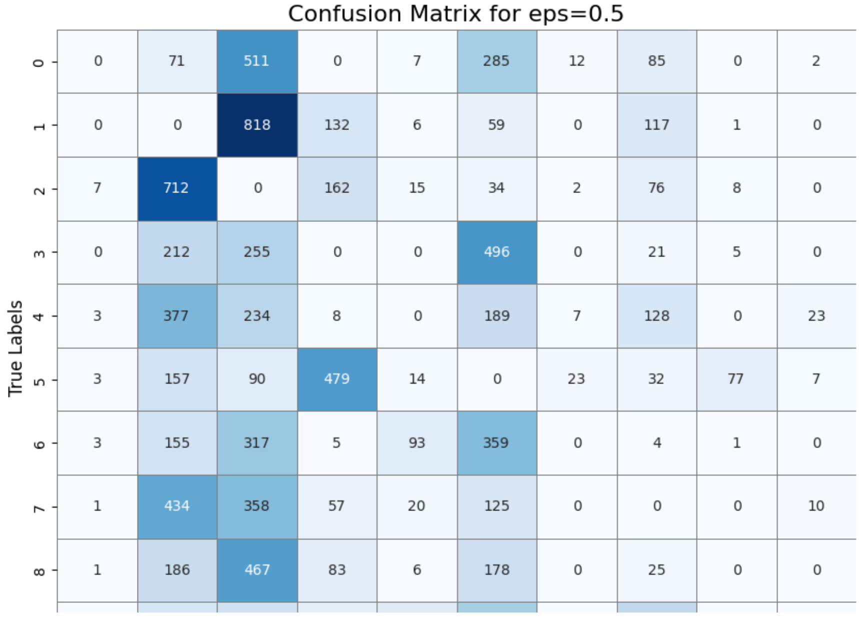

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

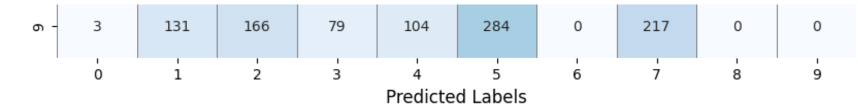
Confusion Matrix for eps=0.4




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```

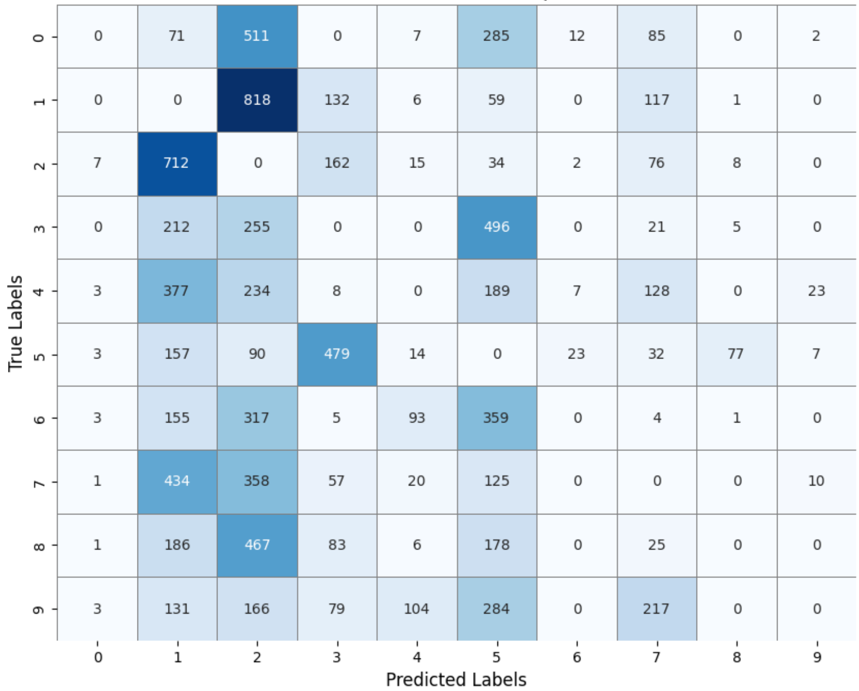
Confusion Matrix for eps=0.5





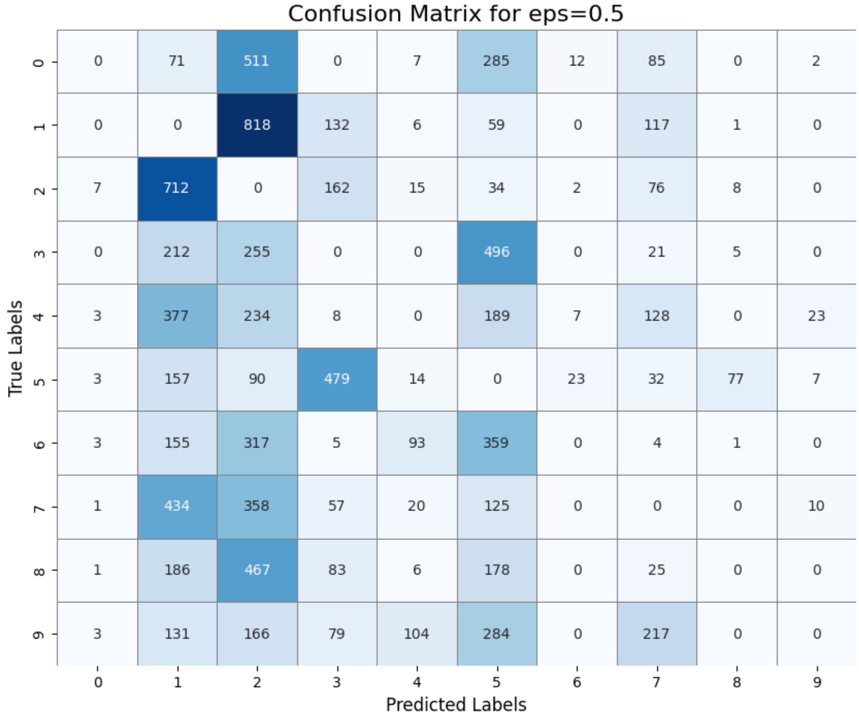

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.5



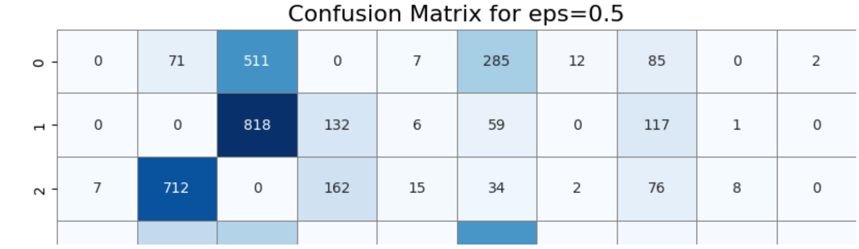

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```

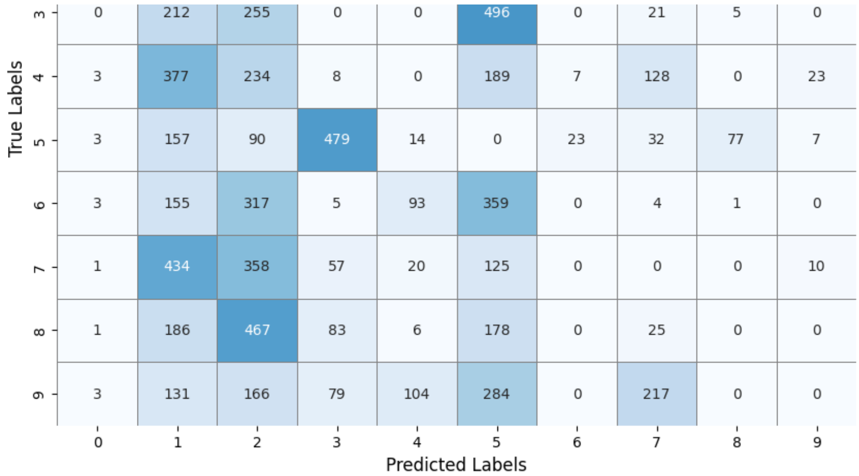
Confusion Matrix for eps=0.5



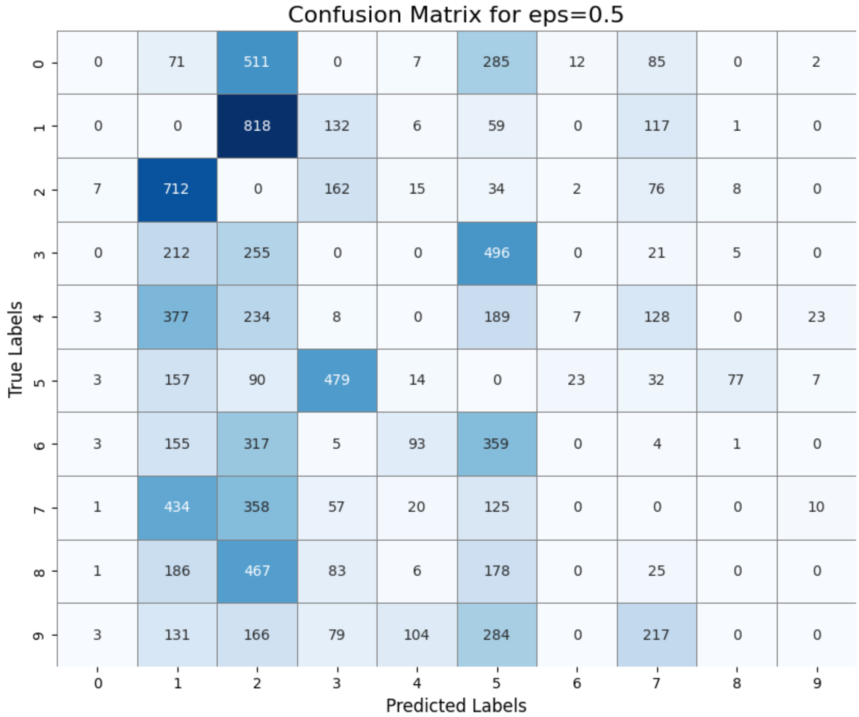

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.5

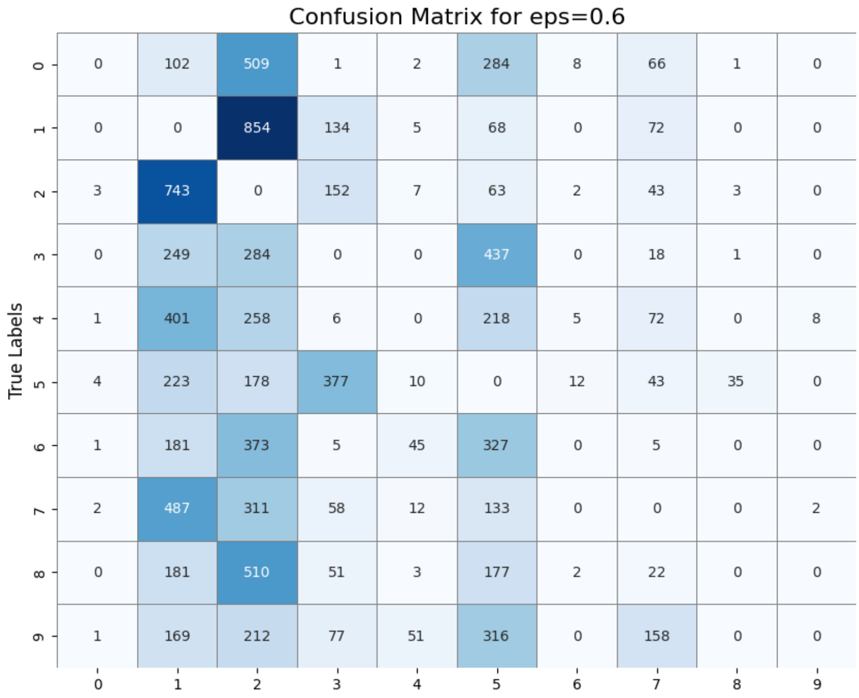





```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```



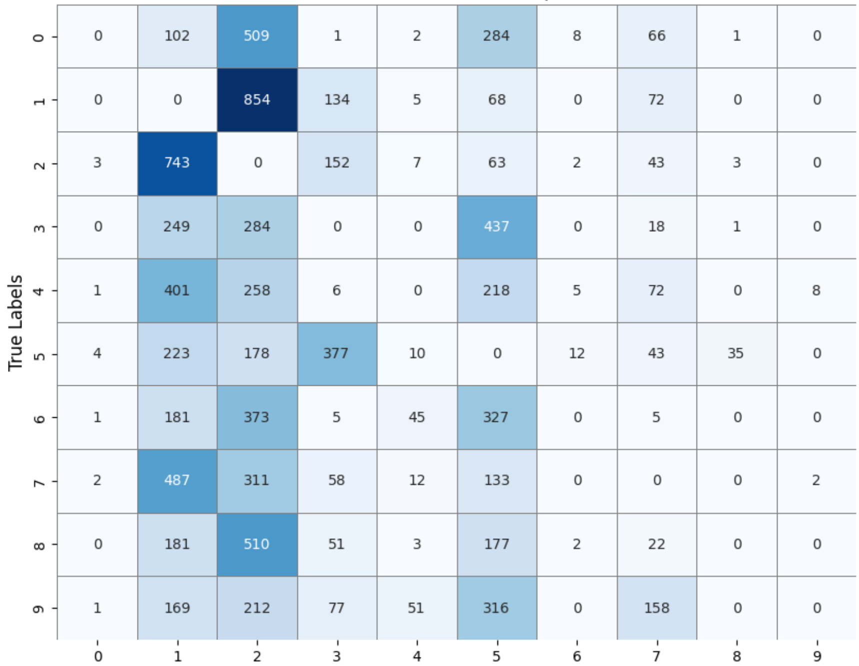

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```



Predicted Labels

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

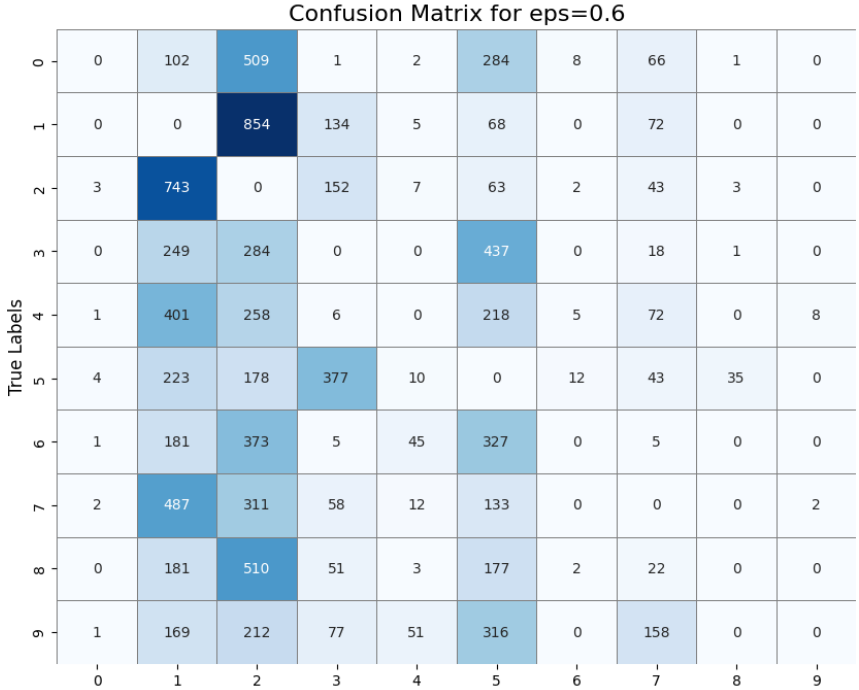
Confusion Matrix for eps=0.6





```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

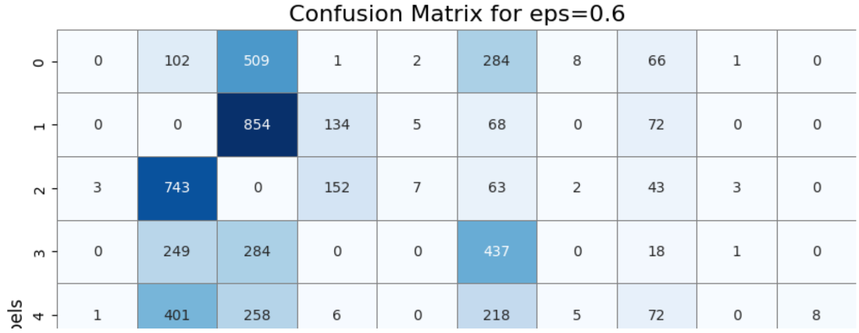
Confusion Matrix for eps=0.6

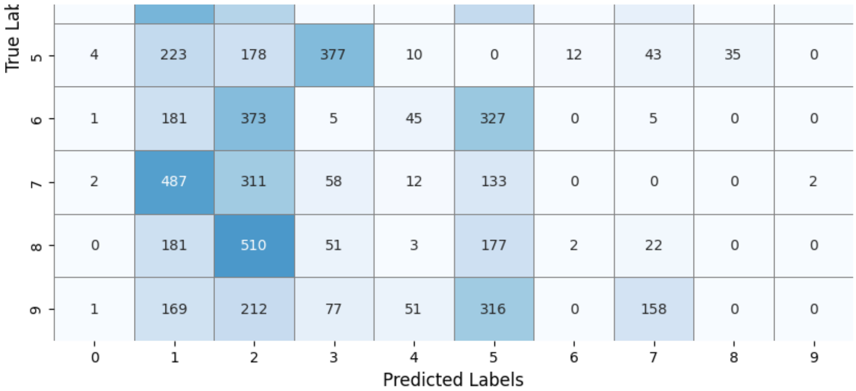





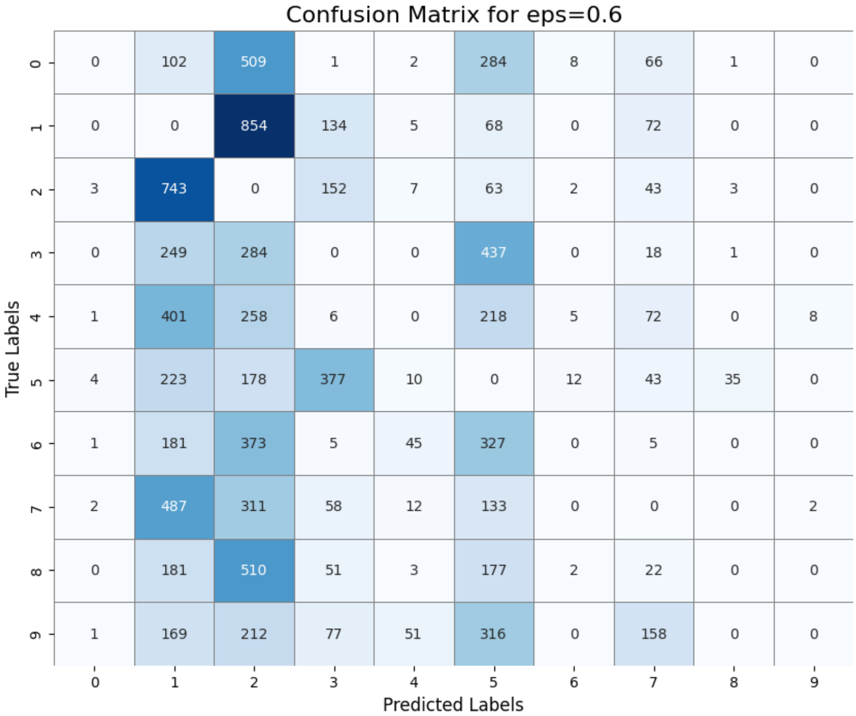
```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.6




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

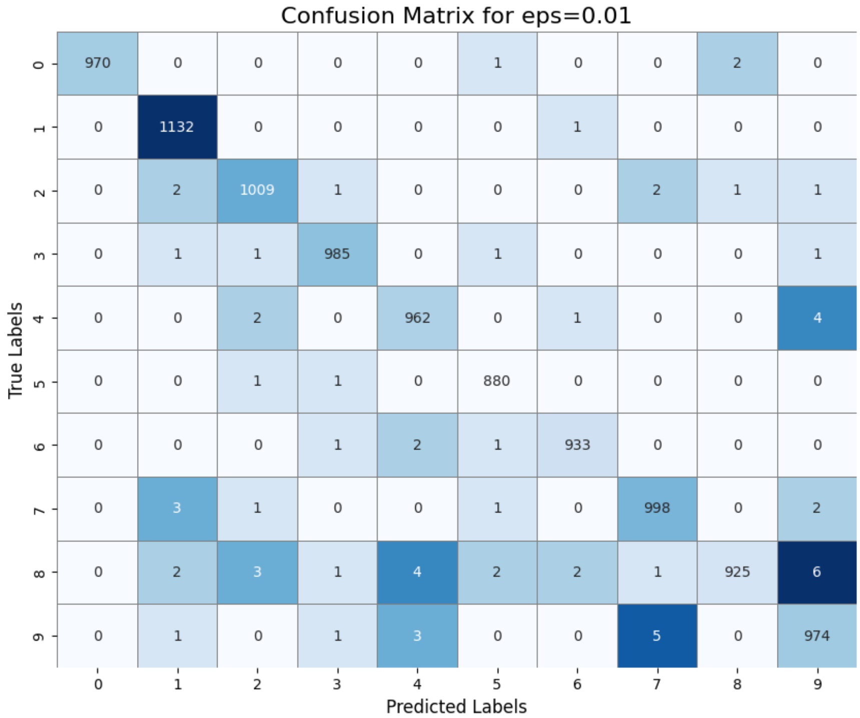


## DeepFoolAttack

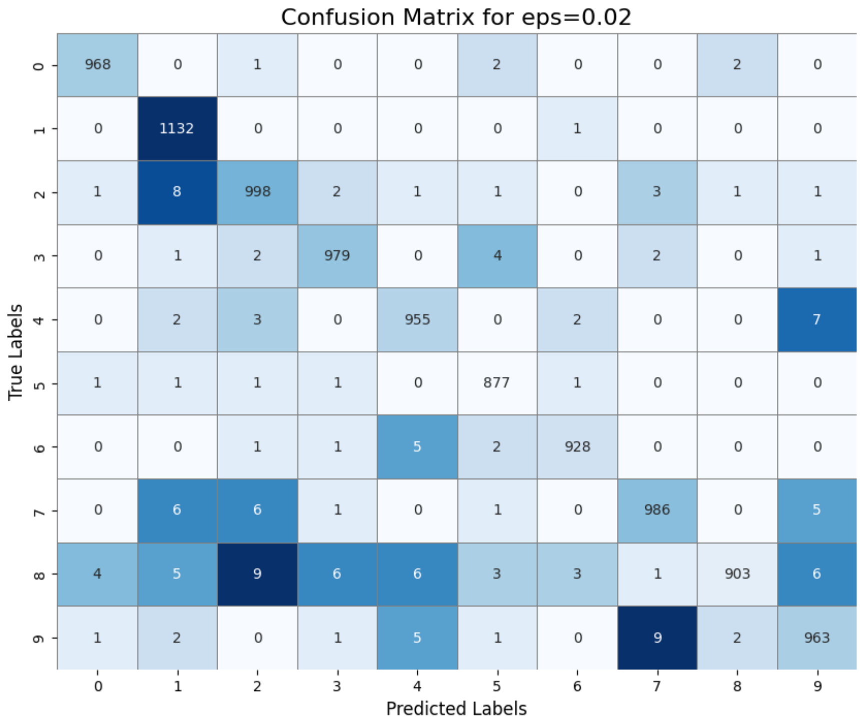
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
import foolbox as fb
import eagerpy as ep
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

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

# Load and preprocess the examples
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy').astype('int32')  # Cast labels to int32

# Create a Foolbox model
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))



# Calculate accuracy on clean data
clean_predictions = model.predict(correct_examples).argmax(axis=-1)
accuracy_clean = np.mean(clean_predictions == correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Convert data to TensorFlow tensors
images = tf.convert_to_tensor(correct_examples)
labels = tf.convert_to_tensor(correct_labels, dtype=tf.int32)  # Cast labels to int32

# Create a Foolbox model for TensorFlow
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

# Apply an FGSM attack
attack = fb.attacks.LinfDeepFoolAttack()
epsilons = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
raw_advs, clipped_advs, success = attack(fmodel, images, labels, epsilons=epsilons)




# Assuming 'clipped_advs' are the adversarial examples for different epsilons
for eps, advs_ in zip(epsilons, clipped_advs):
    # Predict the labels of the adversarial examples
    y_adv = np.argmax(model.predict(advs_), axis=1)

    # Calculate accuracy on adversarial examples
    accuracy_adv = np.mean(y_adv == correct_labels)
    print(f"\nAdversarial 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
    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
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey')

    # 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)

    # Save the plot
    image_filename = f'confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()  # Display the figure in the notebook
```

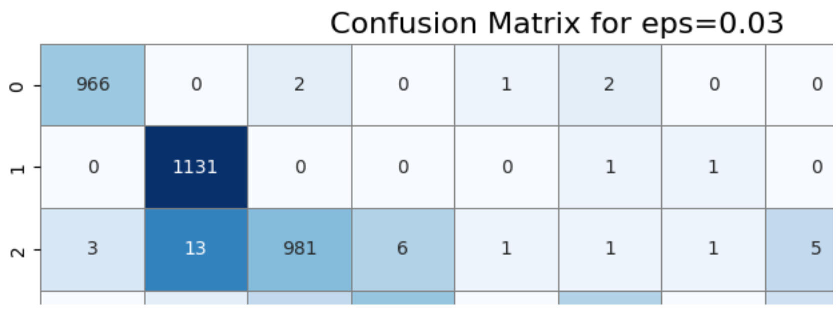
```
308/308 [==============================] - 1s 2ms/step
Accuracy on clean data: 100.00%
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.33%
```




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 98.53%
```

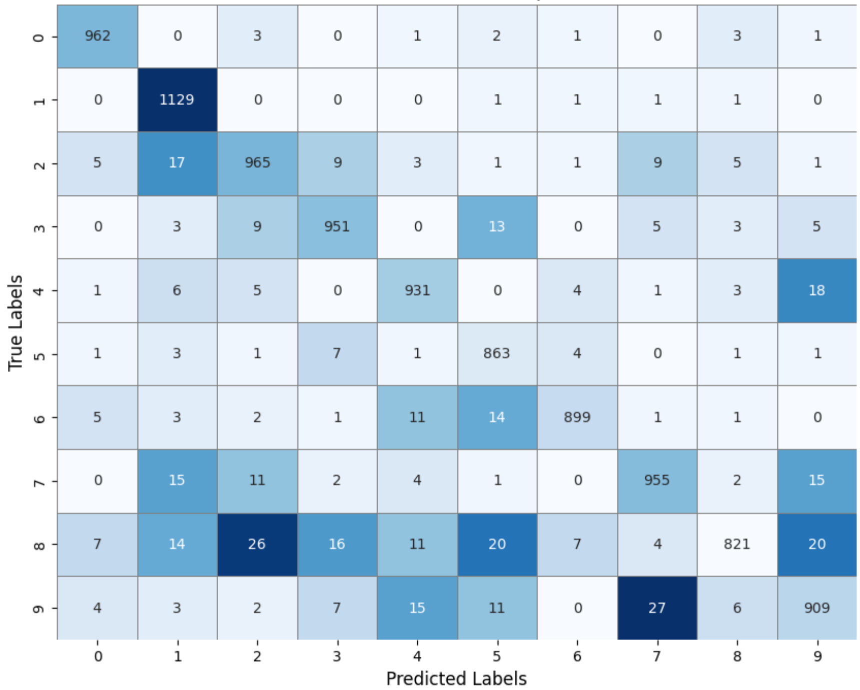



```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 97.06%
```



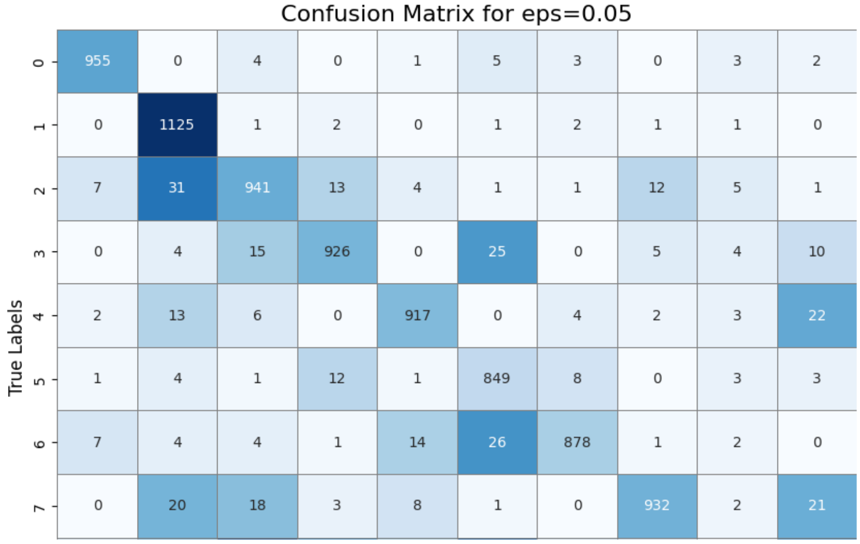

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 95.43%
```

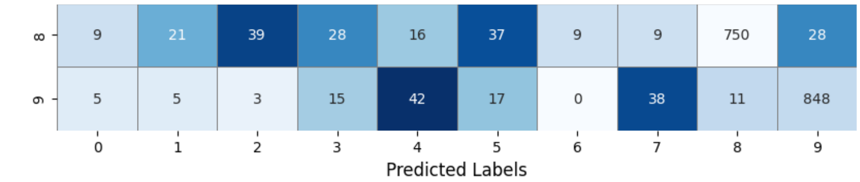
Confusion Matrix for eps=0.04




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 92.75%
```

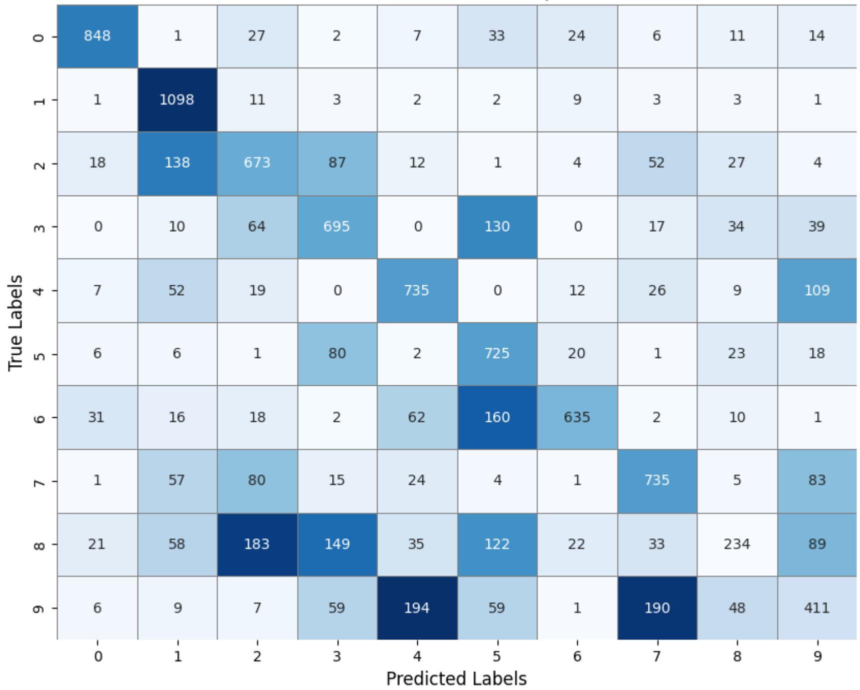
Confusion Matrix for eps=0.05





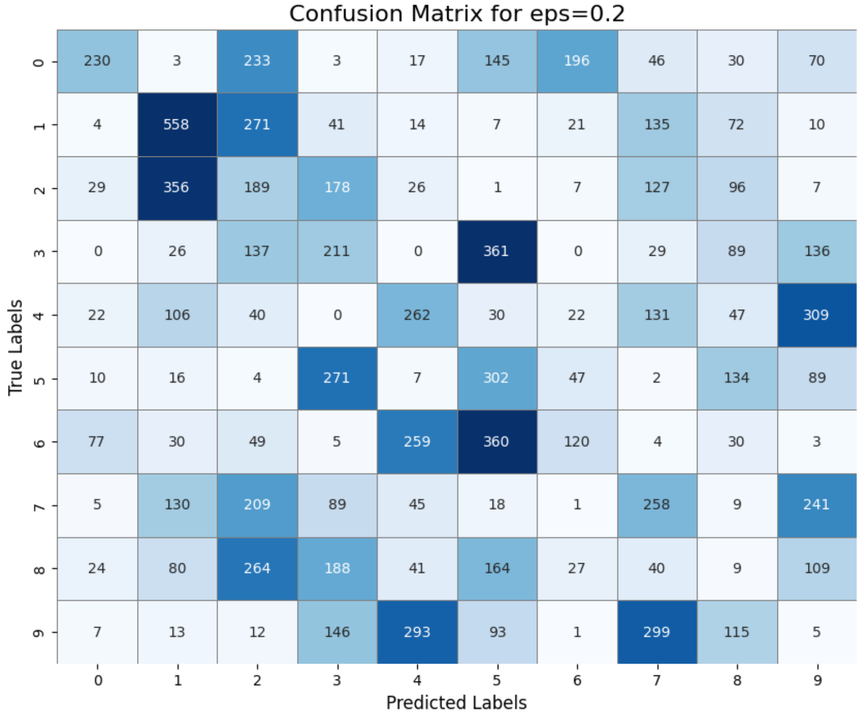

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.1
Accuracy on adversarial examples: 69.04%
```

Confusion Matrix for eps=0.1



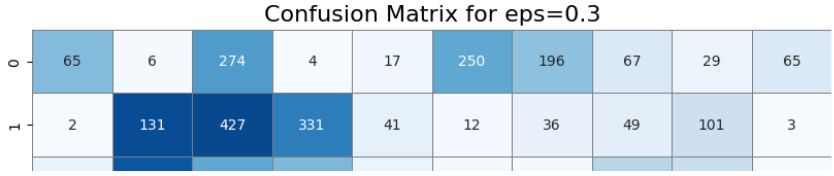

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 21.80%
```

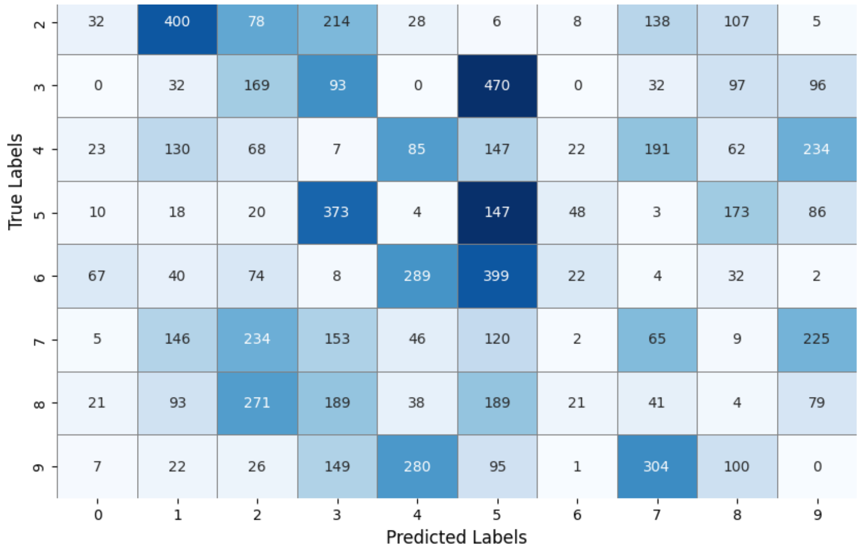
Confusion Matrix for eps=0.2




```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 7.02%
```

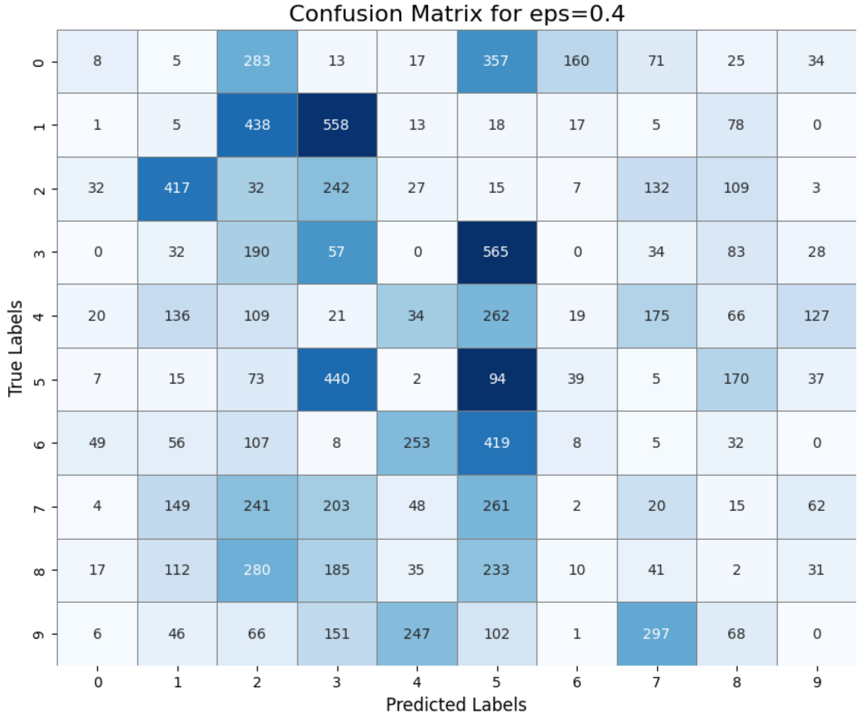
Confusion Matrix for eps=0.3





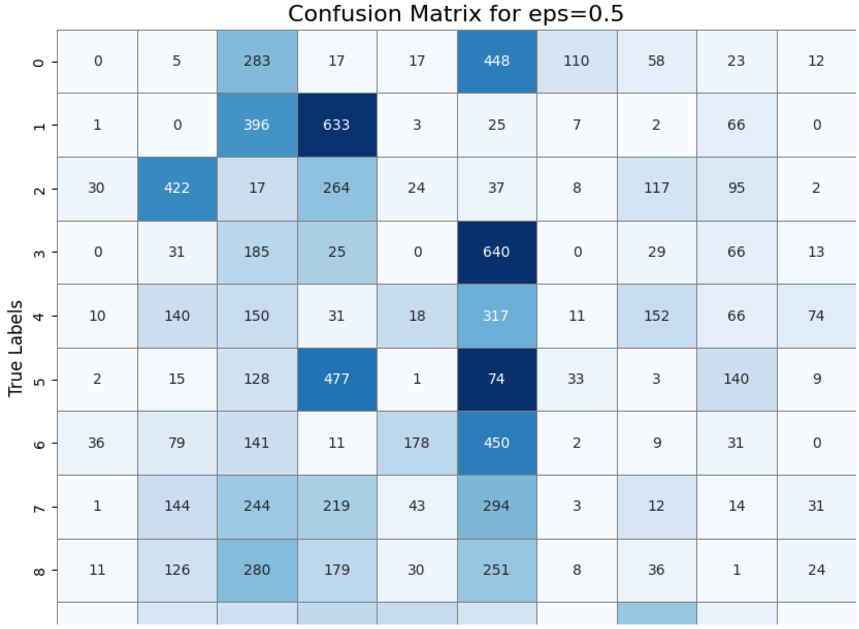

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 2.64%
```

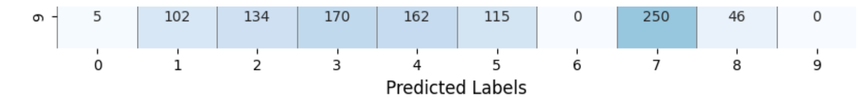
Confusion Matrix for eps=0.4



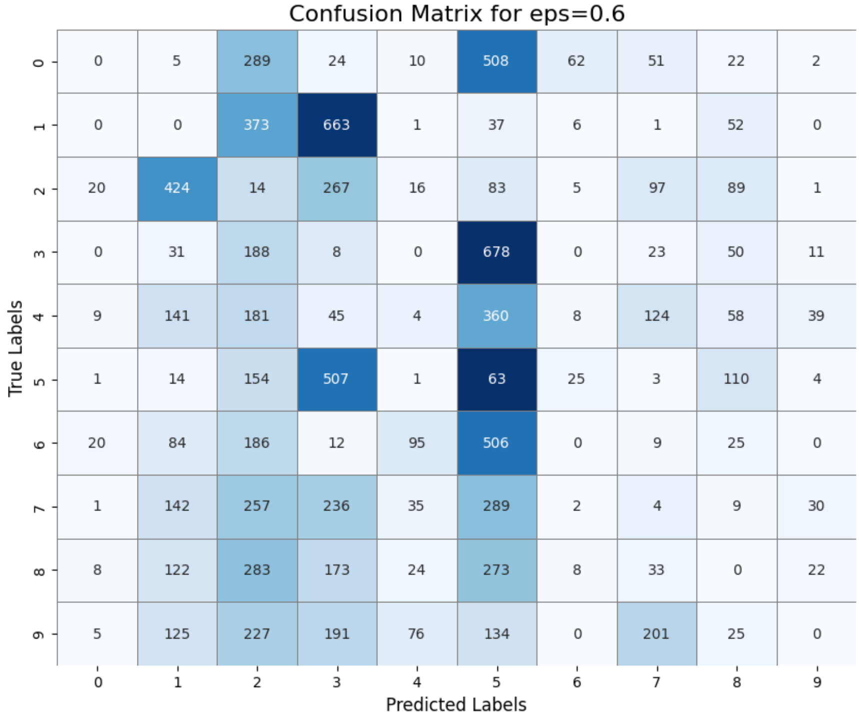

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 1.52%
```

Confusion Matrix for eps=0.5






```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.95%
```

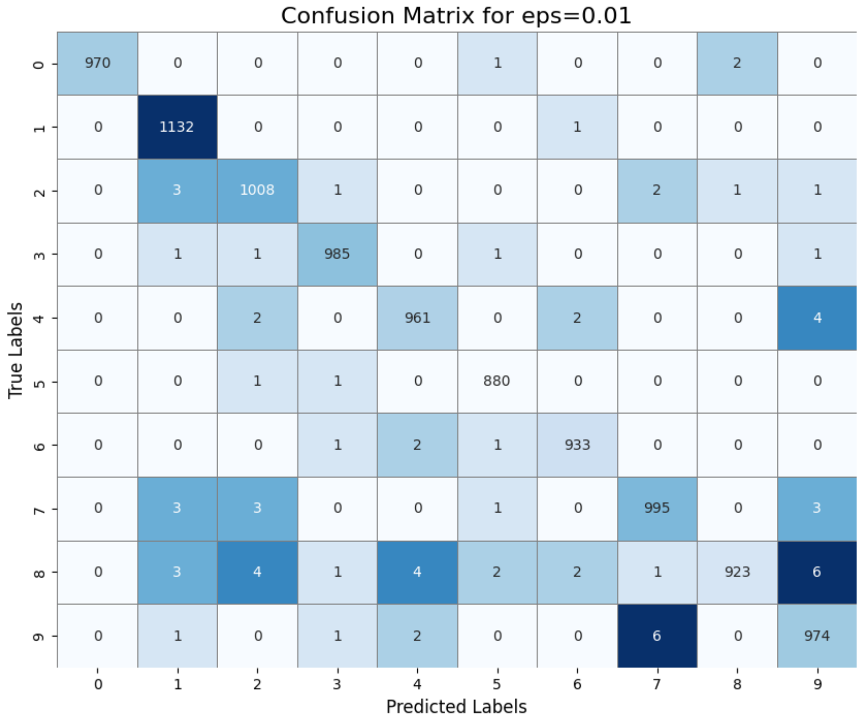



## AdamPGD

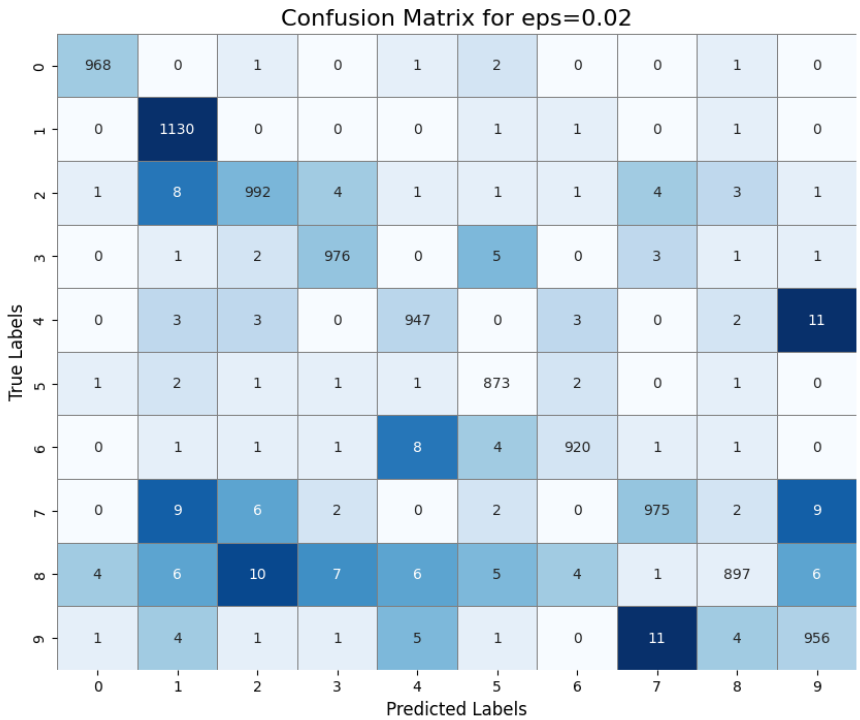
```
import numpy as np
import tensorflow as tf
from keras.models import load_model
import foolbox as fb
import eagerpy as ep
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

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

# Load and preprocess the examples
```

```python
correct_examples = np.load('/content/drive/MyDrive/ColabNotebooks/correct_examples.npy')
correct_labels = np.load('/content/drive/MyDrive/ColabNotebooks/correct_labels.npy').astype('int32')  # Cast labels to int32

# Create a Foolbox model
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))



# Calculate accuracy on clean data
clean_predictions = model.predict(correct_examples).argmax(axis=-1)
accuracy_clean = np.mean(clean_predictions == correct_labels)
print(f"Accuracy on clean data: {accuracy_clean * 100:.2f}%")

# Convert data to TensorFlow tensors
images = tf.convert_to_tensor(correct_examples)
labels = tf.convert_to_tensor(correct_labels, dtype=tf.int32)  # Cast labels to int32

# Create a Foolbox model for TensorFlow
fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

# Apply an FGSM attack
attack = fb.attacks.LinfAdamPGD()
epsilons = [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
raw_advs, clipped_advs, success = attack(fmodel, images, labels, epsilons=epsilons)




# Assuming 'clipped_advs' are the adversarial examples for different epsilons
for eps, advs_ in zip(epsilons, clipped_advs):
    # Predict the labels of the adversarial examples
    y_adv = np.argmax(model.predict(advs_), axis=1)

    # Calculate accuracy on adversarial examples
    accuracy_adv = np.mean(y_adv == correct_labels)
    print(f"\nAdversarial 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
    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
    sns.heatmap(cm, mask=~mask, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False, linewidths=.5, linecolor='grey')

    # 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)

    # Save the plot
    image_filename = f'confusion_matrix_eps_{eps}.png'
    plt.savefig(image_filename, bbox_inches='tight')
    plt.show()  # Display the figure in the notebook
```

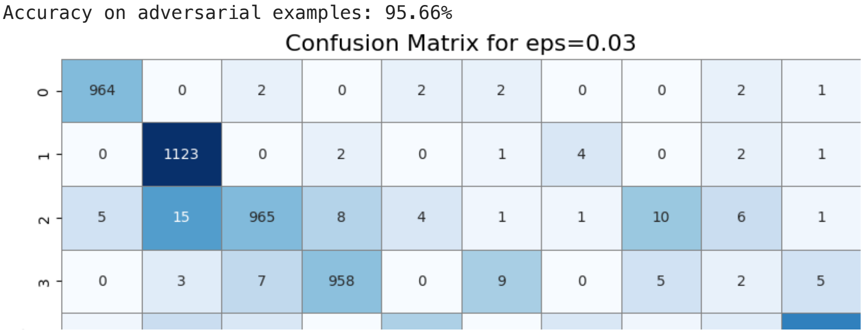
```
308/308 [==============================] - 1s 2ms/step
Accuracy on clean data: 100.00%
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.01
Accuracy on adversarial examples: 99.26%
```

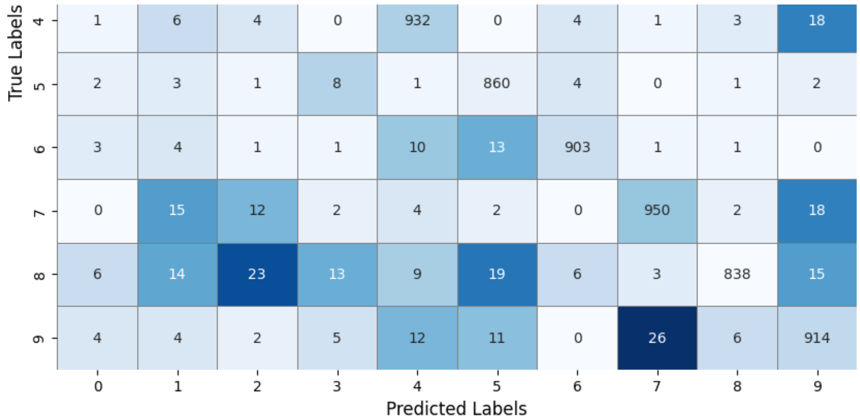



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.02
Accuracy on adversarial examples: 97.97%
```



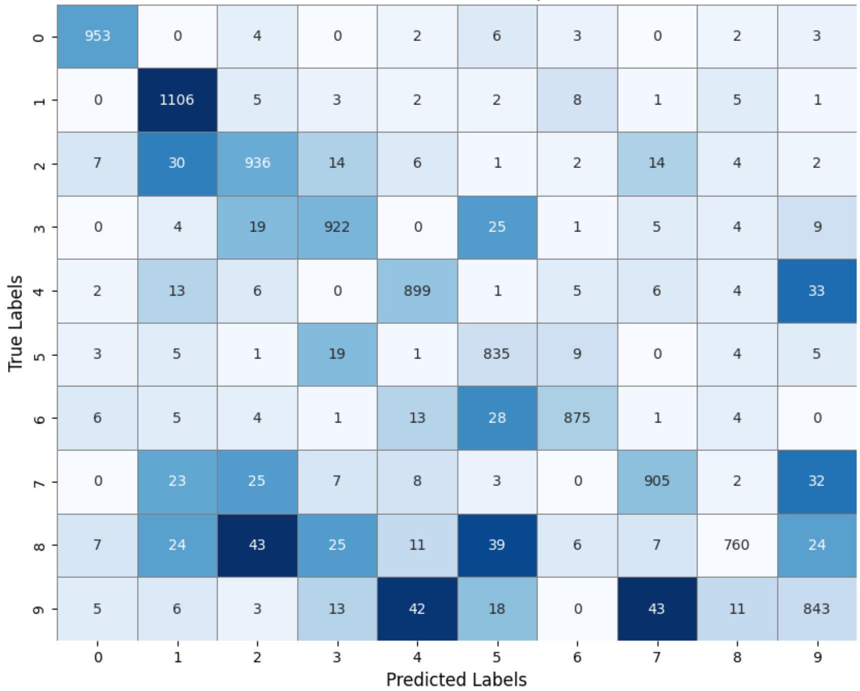

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.03
Accuracy on adversarial examples: 95.66%
```





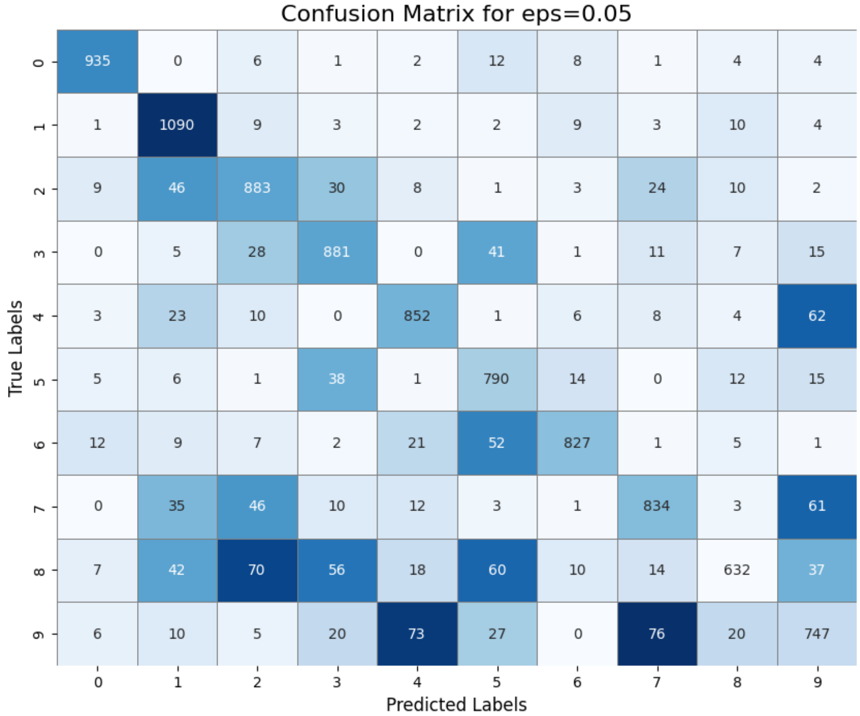

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.04
Accuracy on adversarial examples: 91.86%
```

Confusion Matrix for eps=0.04



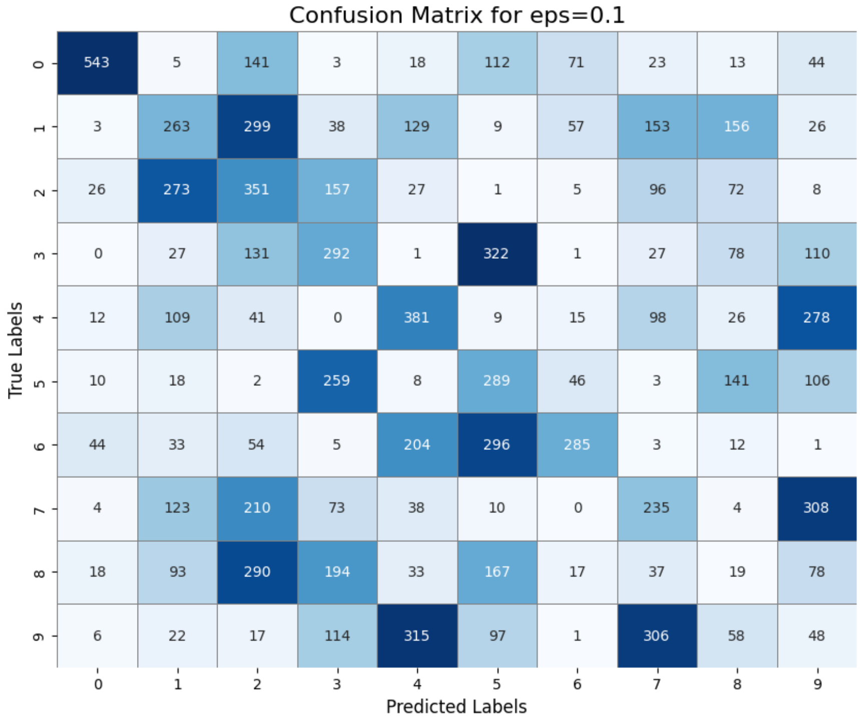

```
308/308 [==============================] - 1s 3ms/step

Adversarial test data: eps:0.05
Accuracy on adversarial examples: 86.14%
```

Confusion Matrix for eps=0.05

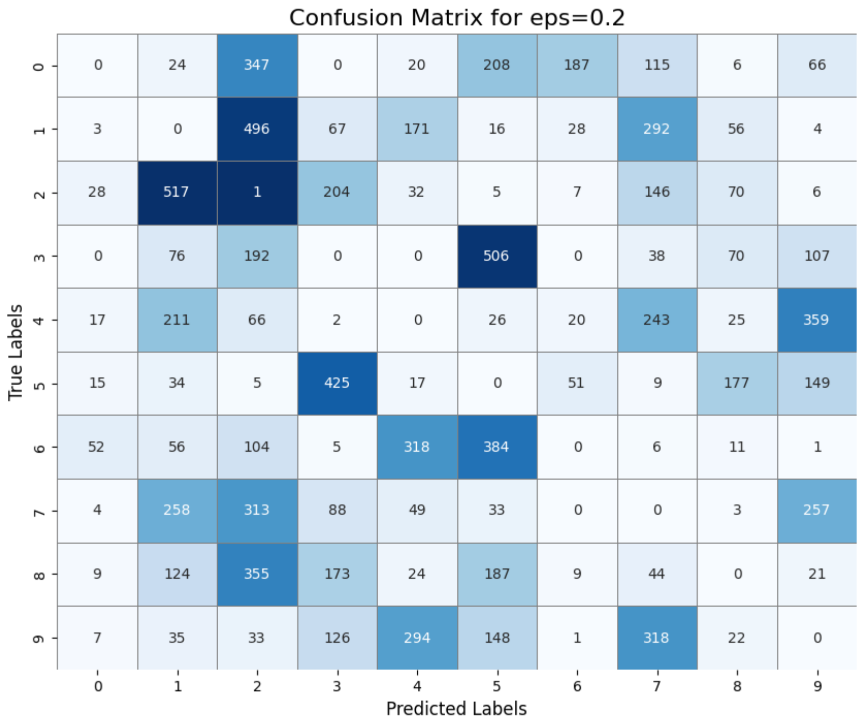



```
308/308 [==============================] - 1s 3ms/step
```

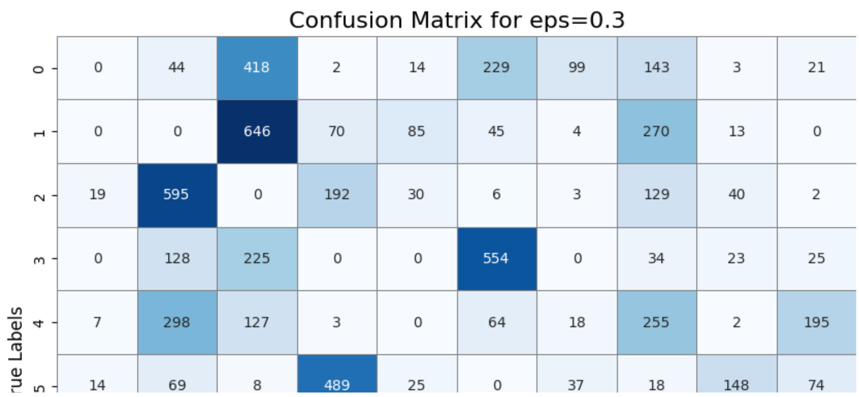
```
Adversarial test data: eps:0.1
Accuracy on adversarial examples: 27.52%
```

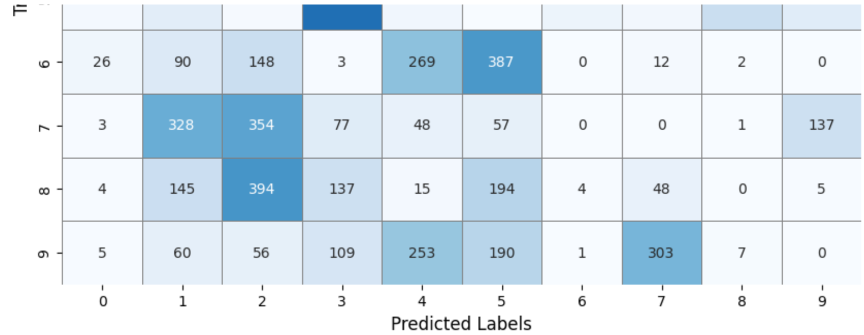



```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.2
Accuracy on adversarial examples: 0.01%
```



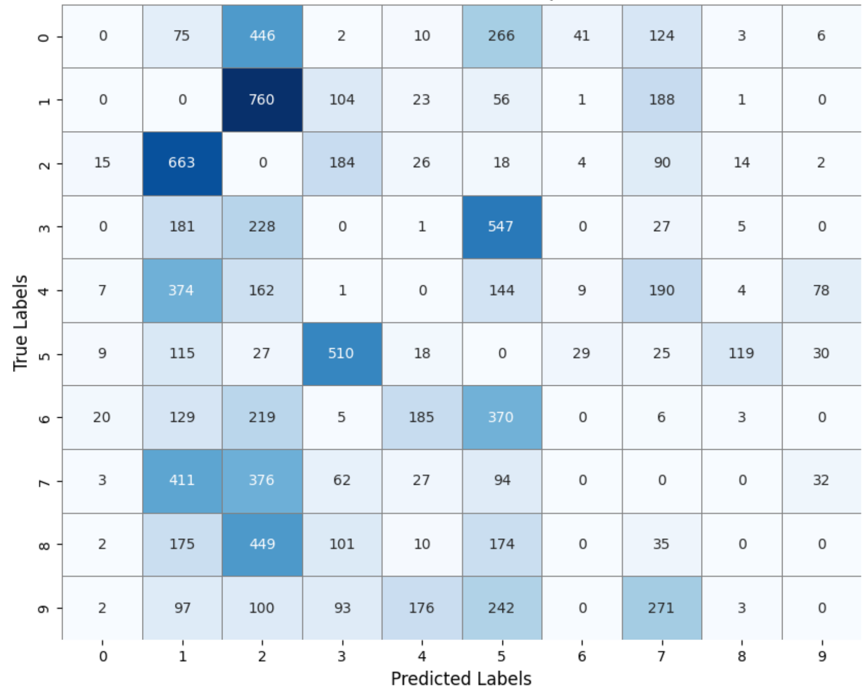

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.3
Accuracy on adversarial examples: 0.00%
```





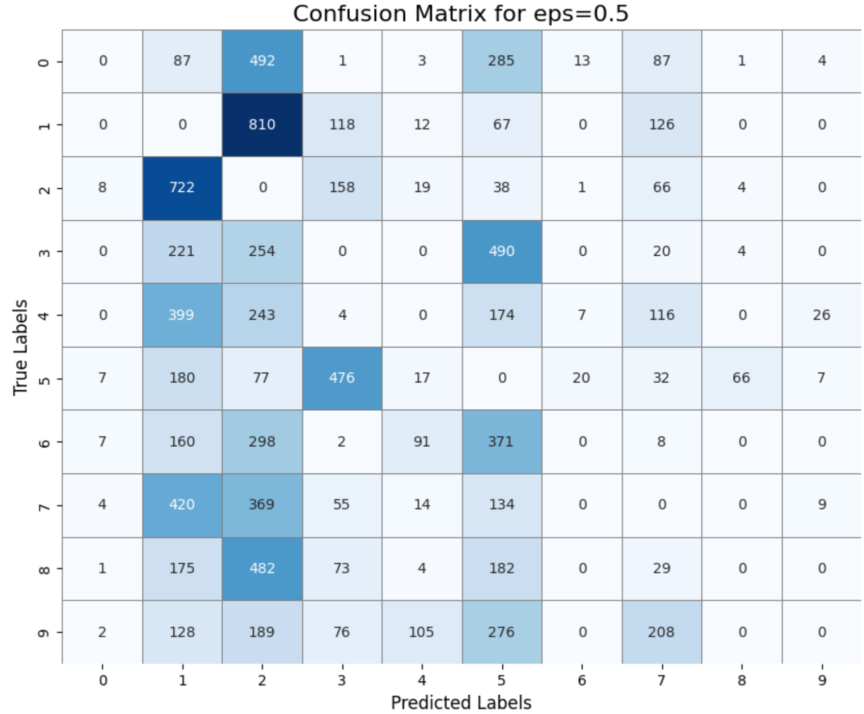

```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.4
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.4




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.5
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.5




```
308/308 [==============================] - 1s 2ms/step

Adversarial test data: eps:0.6
Accuracy on adversarial examples: 0.00%
```

Confusion Matrix for eps=0.6


True Labels
Predicted Labels