Problem 1

In [2]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd
import matplotlib.pyplot as plt
#initialization code required to make tensorflow work on my systemabs
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
#disabling eager execution
tf.compat.v1.disable_eager_execution()
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("Tensorflow version: ",tf.__version__)
```

```
Num GPUs Available:  1
Tensorflow version:  2.4.0
```

In [10]:
```
from art.utils import load_mnist
from art.estimators.classification import KerasClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from art.attacks.evasion import FastGradientMethod
from art.attacks.evasion import CarliniL2Method
from utils import preprocess_mnist
```

As per recommendation on piazza: defining the same architecture as original trained model but with linear activation instead of softmax

In [4]:
```
#define
model = Sequential()
model.add(Conv2D(filters=4, kernel_size=(5, 5), strides=1, activation="relu", input_sha
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=10, kernel_size=(5, 5), strides=1, activation="relu", input_sh
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(100, activation="relu"))
model.add(Dense(10, activation="linear"))
#compile: same as original
model.compile(loss=categorical_crossentropy, optimizer=Adam(learning_rate=0.01), metric
```

In [5]:
```
#load trained weights from original model
model.load_weights(r'mnist_model_weights.h5')
```

In [6]:
```
#save full model for use with Carlini Attacks: no softmax layer only linear with traine
model.save(r'model')
```

```
INFO:tensorflow:Assets written to: model\assets
```

In [11]:
```
#apply ART wrapper: min and max pixel value were 0 and 1 when loaded in using load_mnis
classifier = KerasClassifier(model=model, clip_values=(0.0, 1.0), use_logits=True)
```

Load and preprocessed benign test samples

In [13]:
```
x_attack_test = np.load(r'x_attack_test.npy')
y_attack_test = np.load(r'y_attack_test.npy')
```

In [14]:
```
#define some constants
num_test_samples = x_attack_test.shape[0]
#define target class
t = 7
```

In [15]:
```
num_test_samples
```

Out[15]:
```
100
```

In [16]:
```
#print out accuracy on benign samples
predictions = classifier.predict(x_attack_test)
accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_attack_test, axis=1)) /
print(f"Accuracy on benign test examples: {accuracy*100}%")
```

```
C:\Users\apra\Anaconda3\envs\cy_hw2\lib\site-packages\tensorflow\python\keras\engine\tra
ining.py:2325: UserWarning: `Model.state_updates` will be removed in a future version. T
his property should not be used in TensorFlow 2.0, as `updates` are applied automaticall
y.
  warnings.warn('`Model.state_updates` will be removed in a future version. '
Accuracy on benign test examples: 100.0%
```

In [17]:
```
#print number of target class predictions on benign samples (should be close to 0)
num_targ_class = np.sum(np.argmax(predictions, axis=1) == t) / num_test_samples
```

In [18]:
```
print(f"Percentage predicted to be target class ({t}) on benign test examples: {num_tar
```

```
Percentage predicted to be target class (7) on benign test examples: 0.0%
```

In [19]:
```
x_attack_test.shape
```

Out[19]:
```
(100, 28, 28, 1)
```

In [20]:
```
y_attack_test.shape
```

Out[20]:
```
(100, 10)
```

1a: FSGM attack : we use the softmax model

TARGETTED ATTACKS

In [24]:
```
target_class_vector = t*np.ones(num_test_samples)
```

In [15]:
```
epsilons = [0.1,0.2,0.3,0.4,0.5]
```

In [16]:
```
softmax_model = keras.models.load_model(r'softmax_model')
softmax_classifier = KerasClassifier(model=softmax_model, clip_values=(0.0, 1.0), use_l
```

In [17]:
```
#store best attacks in a list of dictionaries
targeted_fgsm_best_attacks = []
for e in epsilons:
    fsgm_targeted = FastGradientMethod(estimator=softmax_classifier, eps=e,targeted=Tru
    #norm is l_infinity by default
    x_test_fsgm_targeted = fsgm_targeted.generate(x=x_attack_test,y=target_class_vector
    fsgm_targeted_predictions = softmax_classifier.predict(x_test_fsgm_targeted)
    #in the targeted case, attack success rate is simply the percent of test samples cl
    #as target class (since this class is not present in the test set)
    success_mask = np.argmax(fsgm_targeted_predictions, axis=1) == t
    attack_success_rate = np.sum(success_mask) / num_test_samples
    print(f"Attack Success Rate on targeted fsgm attack examples for epsilon = {e} : {r



    #identify and store adversarial inputs for each epsilon with highest target classif
    confidences = pd.Series(fsgm_targeted_predictions[:,7],index=range(num_test_samples

    #get highest successful attacks with highest confidence
    conf_idx = pd.Series.idxmax(confidences[success_mask])

    best_attacks = {'e':e, 'original_image':x_attack_test[conf_idx],
                    'attack_image':x_test_fsgm_targeted[conf_idx],
                    'test_idx' : conf_idx,
                    'perturbation' : (x_attack_test[conf_idx] - x_test_fsgm_targeted[co
    targeted_fsgm_best_attacks.append(best_attacks)
```

```
Attack Success Rate on targeted fsgm attack examples for epsilon = 0.1 : 8.0%
Attack Success Rate on targeted fsgm attack examples for epsilon = 0.2 : 14.0%
Attack Success Rate on targeted fsgm attack examples for epsilon = 0.3 : 24.0%
Attack Success Rate on targeted fsgm attack examples for epsilon = 0.4 : 27.0%
Attack Success Rate on targeted fsgm attack examples for epsilon = 0.5 : 26.0%
```

UNTARGETTED ATTACKS

In [18]:
```
for e in epsilons:
    fsgm_untargeted = FastGradientMethod(estimator=softmax_classifier, eps=e)
    #norm is l_infinity by default
    x_test_fsgm_untargeted = fsgm_untargeted.generate(x=x_attack_test)
    predictions = softmax_classifier.predict(x_test_fsgm_untargeted)
    accuracy_mask = np.argmax(predictions, axis=1) == np.argmax(y_attack_test, axis=1)

    #in untargeted case, attack success rate is the DROP in accuracy i.e, benign_accura
    #this accurately tells us how many previously correctly classified samples were mis
    attack_success_rate = accuracy - np.sum(accuracy_mask) / num_test_samples
    print(f"Attack Success Rate on untargeted fsgm attack examples for epsilon = {e} :
```

```
Attack Success Rate on untargeted fsgm attack examples for epsilon = 0.1 : 34.0%
Attack Success Rate on untargeted fsgm attack examples for epsilon = 0.2 : 53.0%
Attack Success Rate on untargeted fsgm attack examples for epsilon = 0.3 : 63.0%
Attack Success Rate on untargeted fsgm attack examples for epsilon = 0.4 : 69.0%
Attack Success Rate on untargeted fsgm attack examples for epsilon = 0.5 : 75.0%
```

1b Targeted Carlini Wagner in L2: FIXED C

In [65]:

```
constants = [0.5,1,5,10]
```

In [66]:

```
#store attack metrics for various c's in a list of dictionaries
l2_attack_metrics = []
for c in constants:
    #binary search steps is 0 so we can fix the constant c
    carlini = CarliniL2Method(classifier=classifier,targeted=True,initial_const=c,batch
    carlini_attacks = carlini.generate(x_attack_test,target_class_vector)

    #predictions
    carlini_predict = model.predict(carlini_attacks)

    #calculate success rate
    success_mask = np.argmax(carlini_predict, axis=1) == t
    attack_success_rate = sum(success_mask)/num_test_samples

    #calculate l2 norm
    l2dist = np.sqrt(np.sum(np.square(x_attack_test - carlini_attacks).reshape(num_test
    #create series out of l2 dist to pull out min,max,median
    l2dist = pd.Series(l2dist,index=range(num_test_samples))

    #Report attack success rate and avg l2 norm
    print(f"Attack Success Rate on targeted Carlini Wagner attack examples for c = {c}
    print(f"Avg L2 Norm on targeted Carlini Wagner attack examples for c = {c} : {round


    min_l2_idx = pd.Series.idxmin(l2dist[success_mask])
    max_l2_idx = pd.Series.idxmax(l2dist[success_mask])
    median_l2_idx = int(l2dist[success_mask].sort_values(ignore_index=False).reset_inde

    #append the l2 metrics
    l2_attack_metrics.append(
    {
        'c':c,
        'min_l2_idx':min_l2_idx,
        'max_l2_idx':max_l2_idx,
        'median_l2_idx':median_l2_idx,

        'min_l2':l2dist[min_l2_idx],
        'max_l2':l2dist[max_l2_idx],
        'median_l2':l2dist[median_l2_idx],

        'original_image_min_l2':x_attack_test[min_l2_idx],
        'original_image_max_l2':x_attack_test[max_l2_idx],
        'original_image_median_l2':x_attack_test[median_l2_idx],

        'attack_image_min_l2':carlini_attacks[min_l2_idx],
        'attack_image_max_l2':carlini_attacks[max_l2_idx],
        'attack_image_median_l2':carlini_attacks[median_l2_idx],
```

```
        'perturbation_min_l2':x_attack_test[min_l2_idx]- carlini_attacks[min_l2_idx],
        'perturbation_max_l2':x_attack_test[max_l2_idx] - carlini_attacks[max_l2_idx],
        'perturbation_median_l2':x_attack_test[median_l2_idx] - carlini_attacks[median_




    })
```

```
Attack Success Rate on targeted Carlini Wagner attack examples for c = 0.5 : 14.0%
Avg L2 Norm on targeted Carlini Wagner attack examples for c = 0.5 : 0.22

Attack Success Rate on targeted Carlini Wagner attack examples for c = 1 : 25.0%
Avg L2 Norm on targeted Carlini Wagner attack examples for c = 1 : 0.52

Attack Success Rate on targeted Carlini Wagner attack examples for c = 5 : 60.0%
Avg L2 Norm on targeted Carlini Wagner attack examples for c = 5 : 2.25

Attack Success Rate on targeted Carlini Wagner attack examples for c = 10 : 71.0%
Avg L2 Norm on targeted Carlini Wagner attack examples for c = 10 : 3.14
```

Observations:

1. As we increase c, the attack success rate seems to increase steadily.
2. As we increase c, the avg l2 perturbations also seems to increase. We can also observe that the approximately linear. Doubling c doubles avg l2 and 5x'ing c 5'xs the avg l2 perturbation.

1c Vizualizations

(Successful) Adversarial samples with highest confidence for targeted FSGM

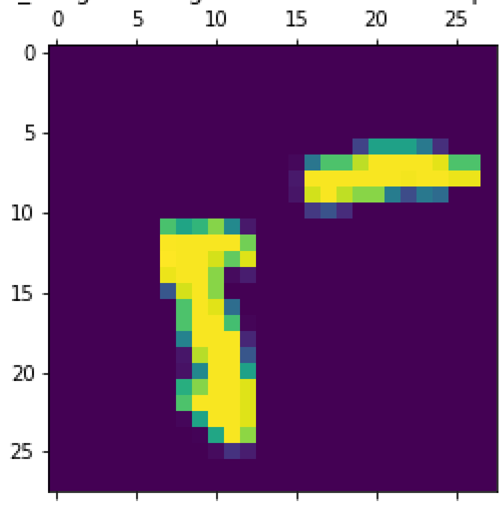
We extracted this information in our attack loop

In [54]:

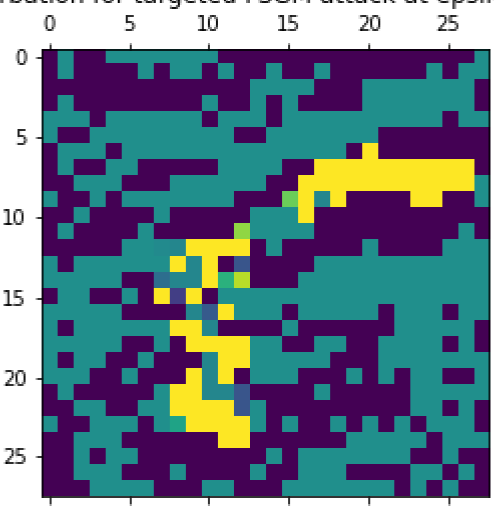
```
for e_idx,e in enumerate(epsilons):
    print(f"==========EPSILON = {e}==============")
    viz_type = "original_image"
    plt.matshow(targeted_fsgm_best_attacks[e_idx][viz_type])
    plt.title(f"{viz_type} for targeted FSGM attack at epsilon = {epsilons[e_idx]}")
    plt.show()
    viz_type = "perturbation"
    plt.matshow(targeted_fsgm_best_attacks[e_idx][viz_type])
    plt.title(f"{viz_type} for targeted FSGM attack at epsilon = {epsilons[e_idx]}")
    plt.show()
    viz_type = "attack_image"
    plt.matshow(targeted_fsgm_best_attacks[e_idx][viz_type])
    plt.title(f"{viz_type} for targeted FSGM attack at epsilon = {epsilons[e_idx]}")
    plt.show()
    print()
```

```
==========EPSILON = 0.1==============
```

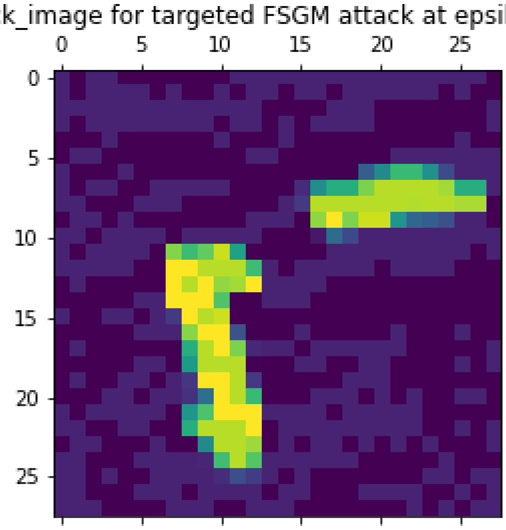
original_image for targeted FSGM attack at epsilon = 0.1



perturbation for targeted FSGM attack at epsilon = 0.1

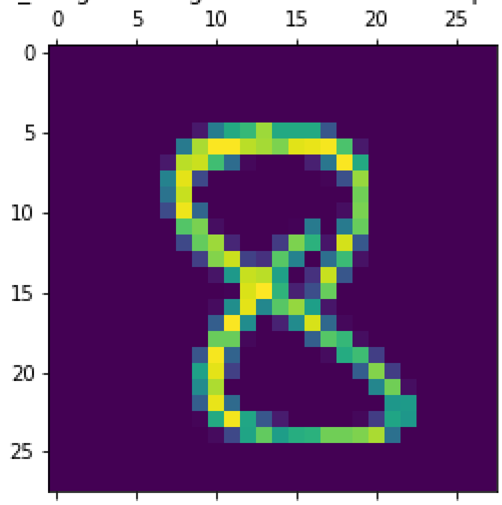


attack_image for targeted FSGM attack at epsilon = 0.1

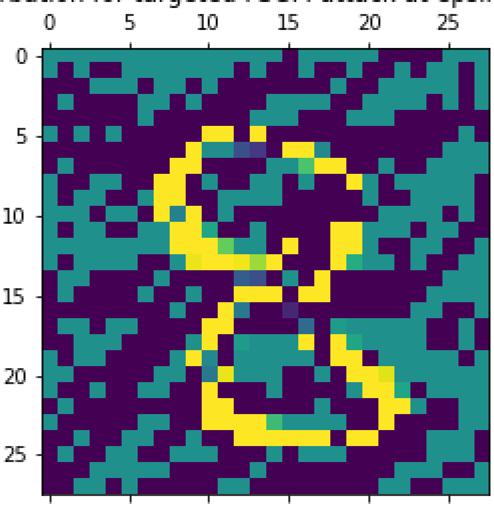



```
==========EPSILON = 0.2==============
```

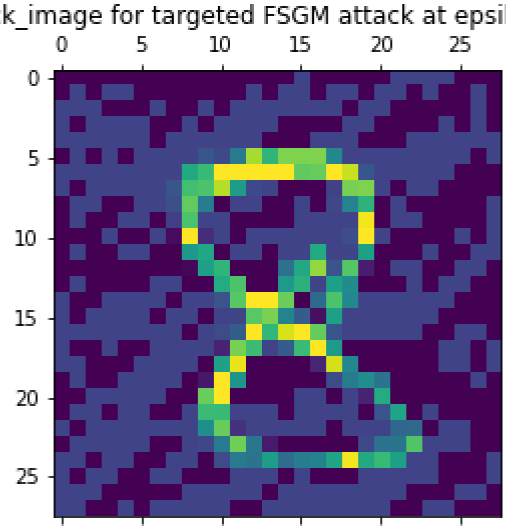
original_image for targeted FSGM attack at epsilon = 0.2



perturbation for targeted FSGM attack at epsilon = 0.2

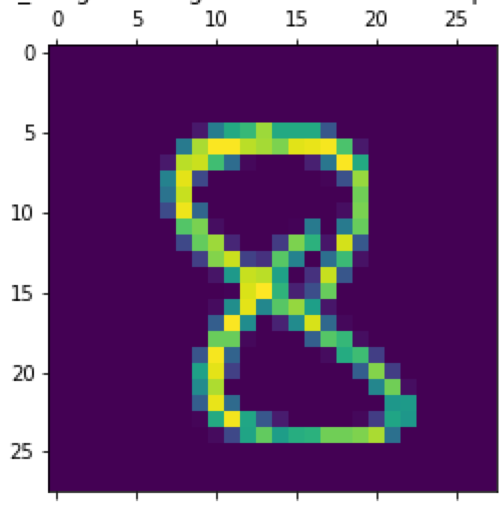


attack_image for targeted FSGM attack at epsilon = 0.2

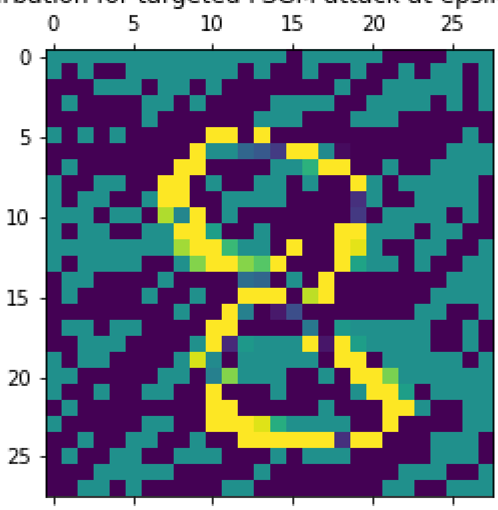


```
==========EPSILON = 0.3==============
```

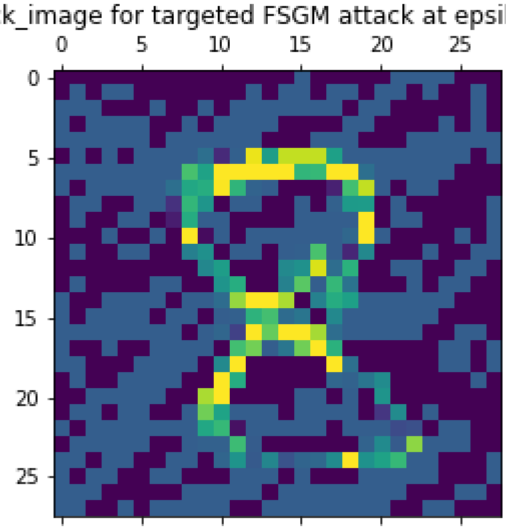
original_image for targeted FSGM attack at epsilon = 0.3



perturbation for targeted FSGM attack at epsilon = 0.3

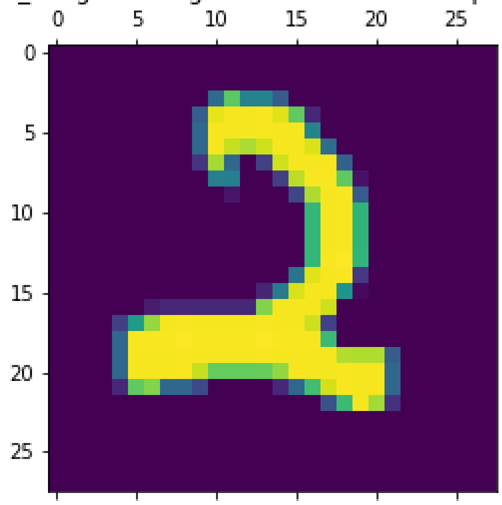


attack_image for targeted FSGM attack at epsilon = 0.3

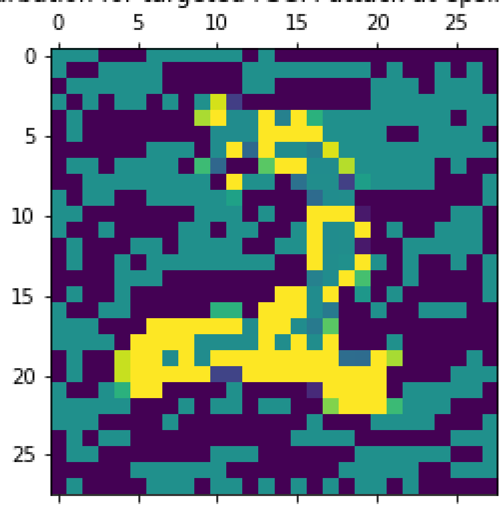



```
==========EPSILON = 0.4==============
```

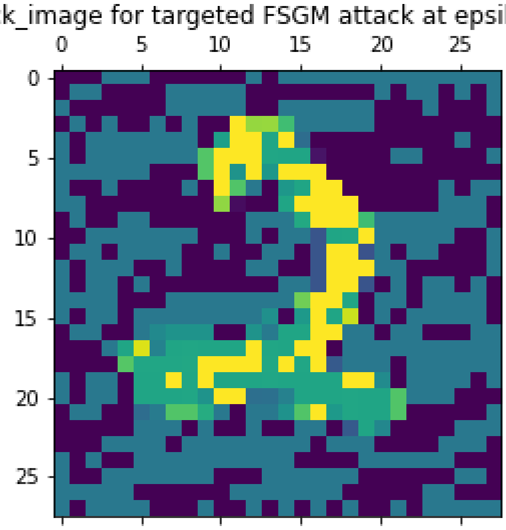
original_image for targeted FSGM attack at epsilon = 0.4



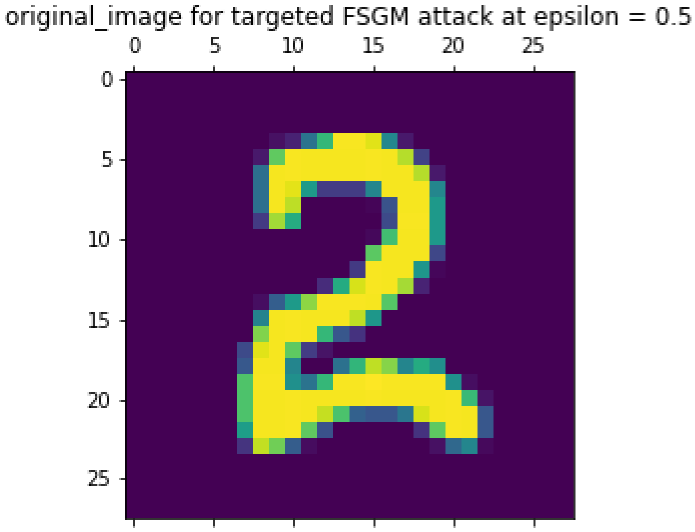
perturbation for targeted FSGM attack at epsilon = 0.4

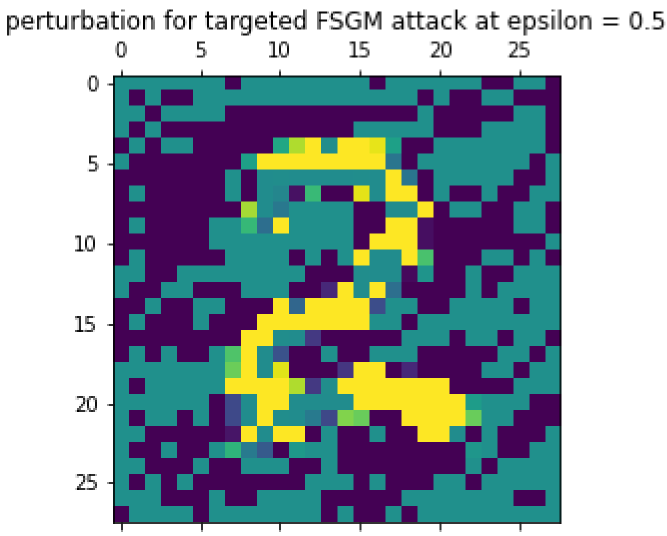


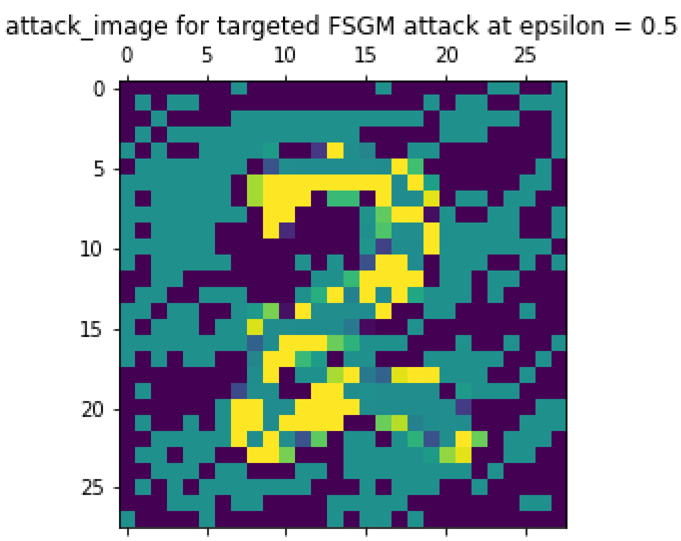
attack_image for targeted FSGM attack at epsilon = 0.4



```
==========EPSILON = 0.5==============
```







Min,Max and Median l2 distance images for various values of c: Successful Carlini Wagner attacks

In [67]:
```
l2_attack_metrics_df = pd.DataFrame(l2_attack_metrics)
```

In [68]:
```
l2_attack_metrics_df[['c','min_l2_idx','min_l2','max_l2_idx','max_l2','median_l2_idx','
```

Out[68]:

	c	min_l2_idx	min_l2	max_l2_idx	max_l2	median_l2_idx	median_l2
0	0.5	55	1.170726	73	2.606154	26	1.584509
1	1.0	74	1.261963	98	3.745318	68	2.126723
2	5.0	55	1.706915	62	7.120429	90	3.563080
3	10.0	26	1.858502	18	7.124496	29	4.463604

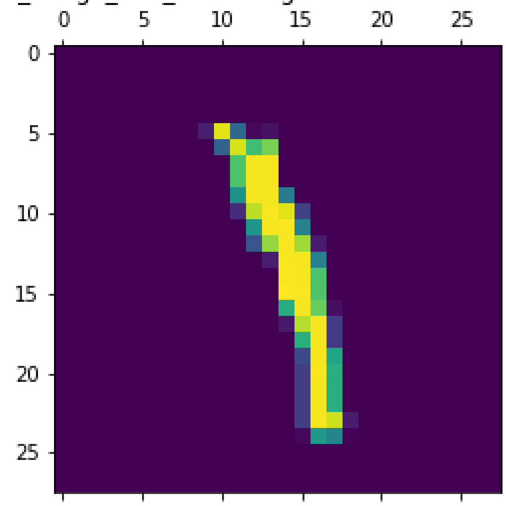
C = 0.5

In [73]:
```
c_i,c = 0,0.5
print(f"==========MIN L2 ================")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
```

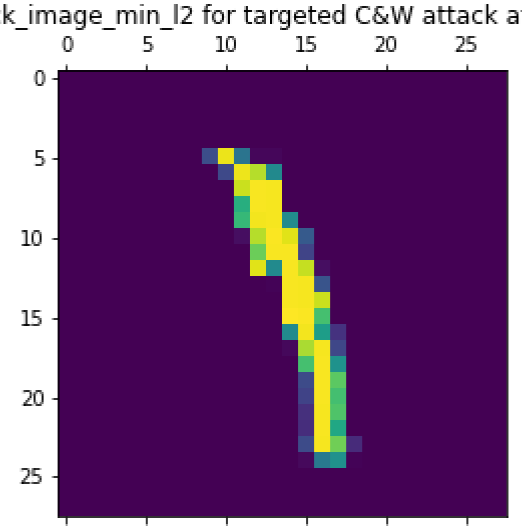
```
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

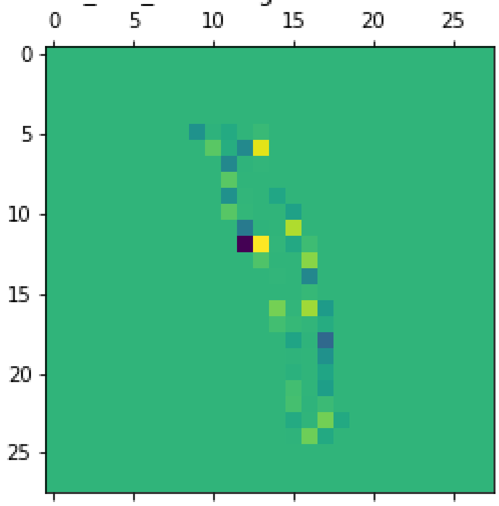
original_image_min_l2 for targeted C&W attack at c = 0.5



attack_image_min_l2 for targeted C&W attack at c = 0.5

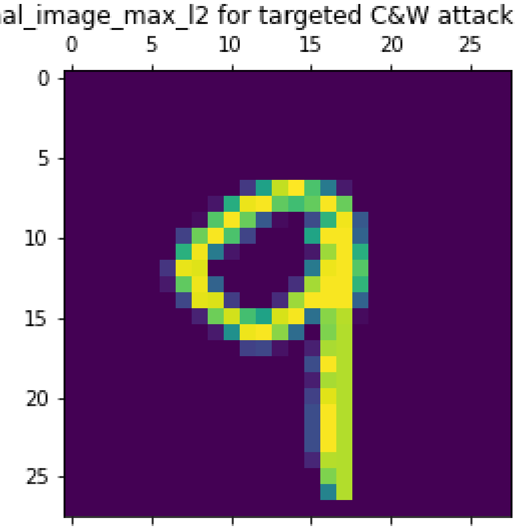


perturbation_min_l2 for targeted C&W attack at c = 0.5

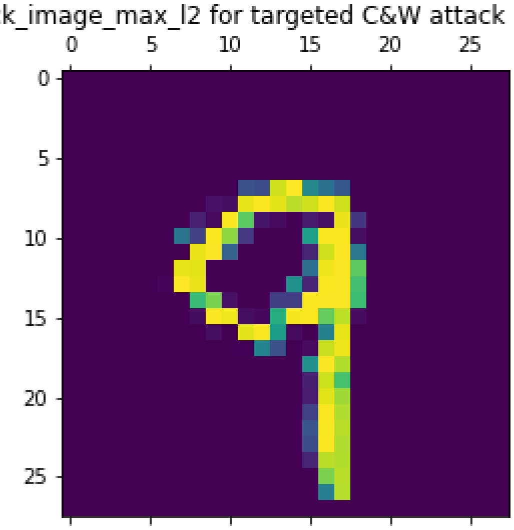


===========MAX L2 ================

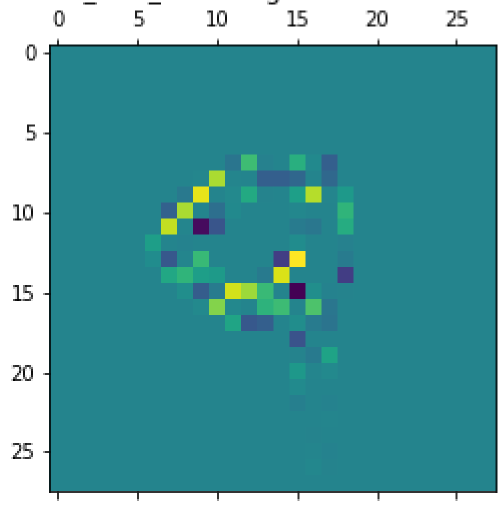
original_image_max_l2 for targeted C&W attack at c = 0.5



attack_image_max_l2 for targeted C&W attack at c = 0.5

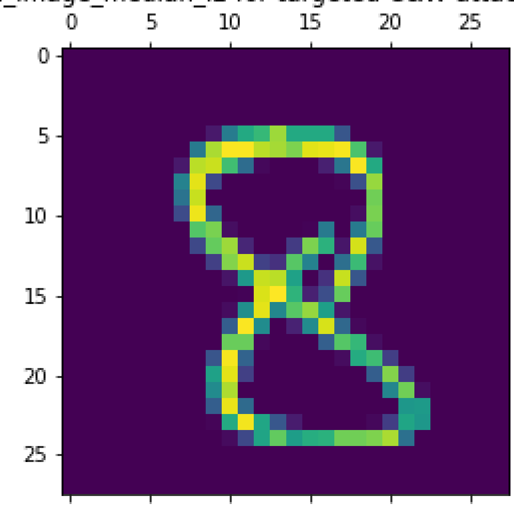


perturbation_max_l2 for targeted C&W attack at c = 0.5

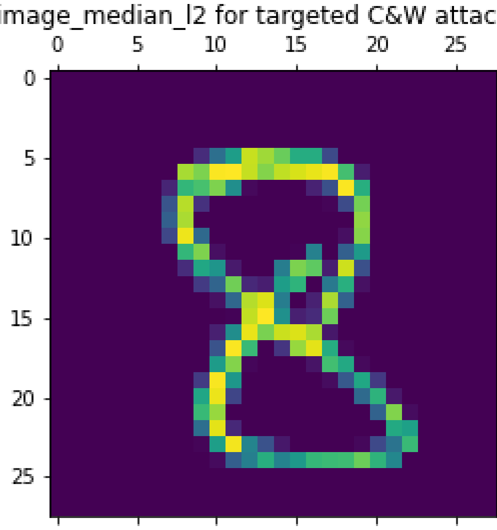


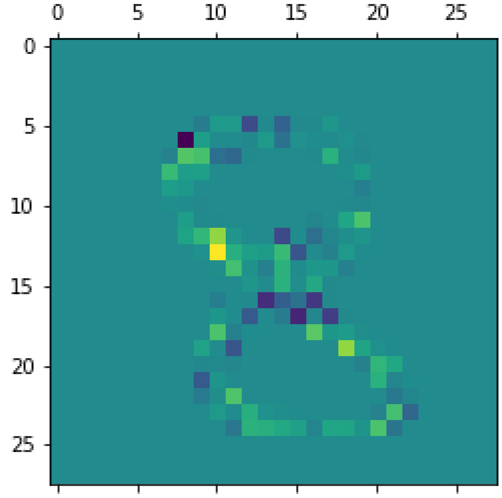

```
===========MEDIAN L2 ================
```

original_image_median_l2 for targeted C&W attack at c = 0.5



attack_image_median_l2 for targeted C&W attack at c = 0.5





C = 1

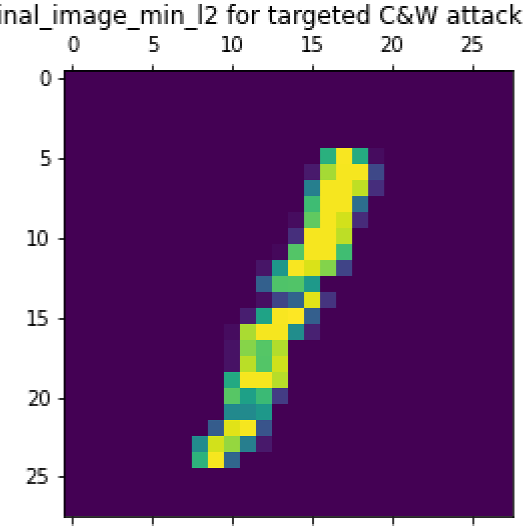
In [74]:

```
c_i,c = 1,1
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ===============")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
```

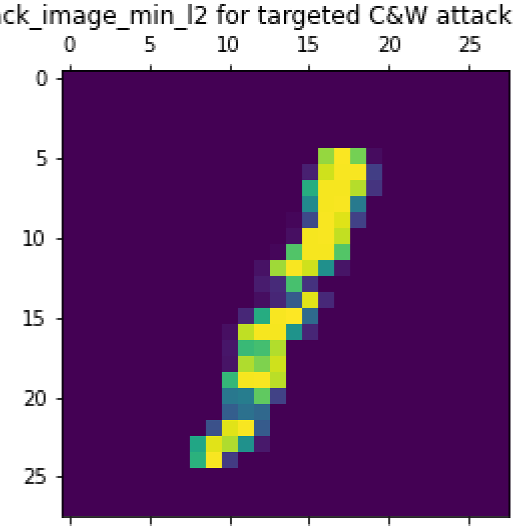
```
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

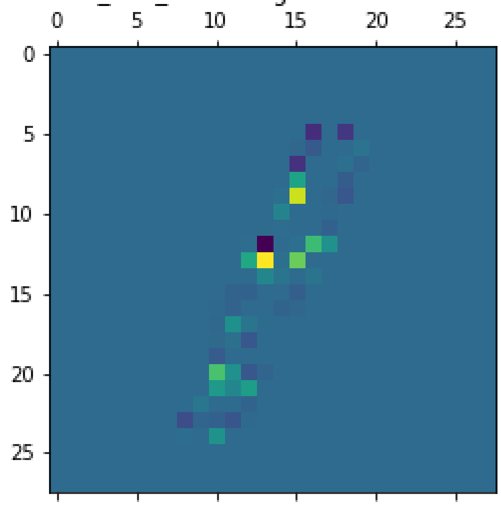
original_image_min_l2 for targeted C&W attack at c = 1



attack_image_min_l2 for targeted C&W attack at c = 1

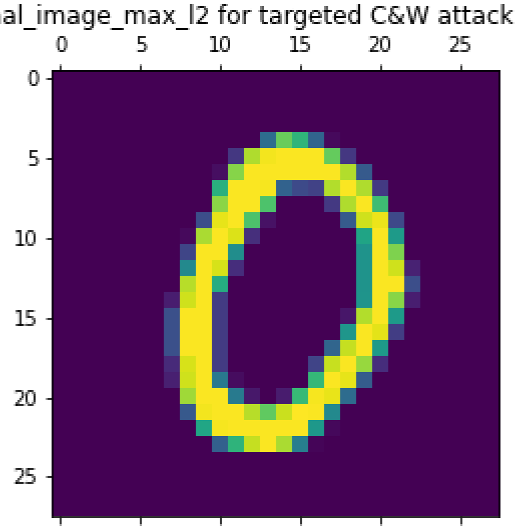


perturbation_min_l2 for targeted C&W attack at c = 1

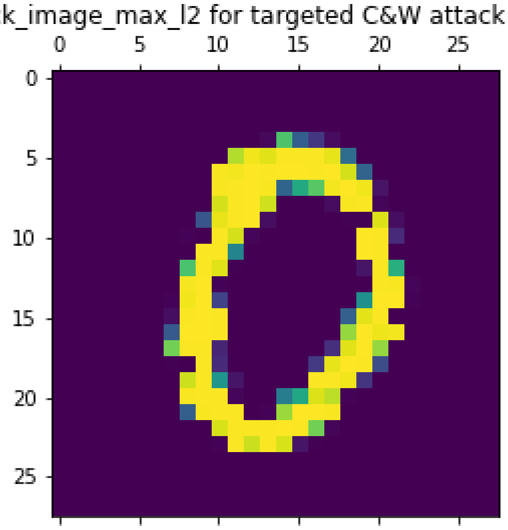


```
===========MAX L2 ================
```

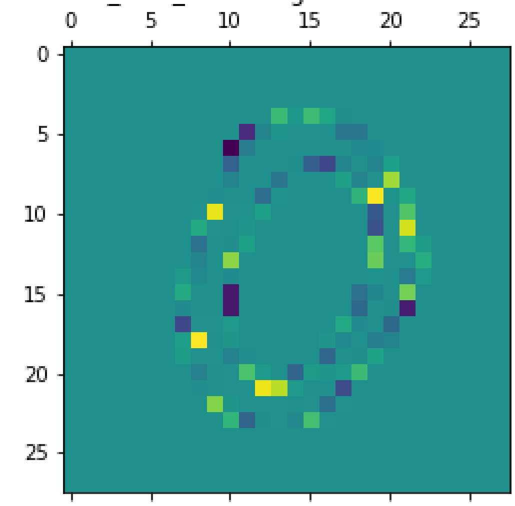
original_image_max_l2 for targeted C&W attack at c = 1



attack_image_max_l2 for targeted C&W attack at c = 1

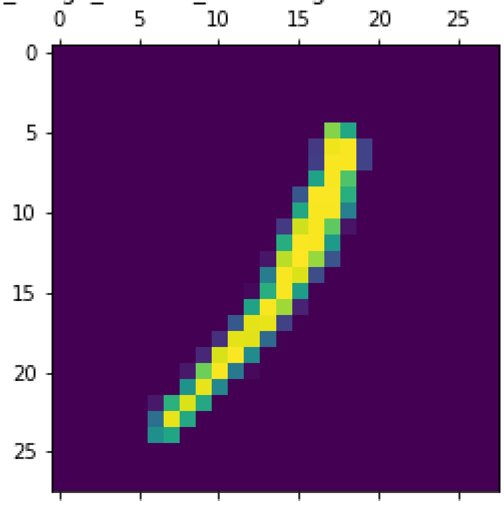


perturbation_max_l2 for targeted C&W attack at c = 1

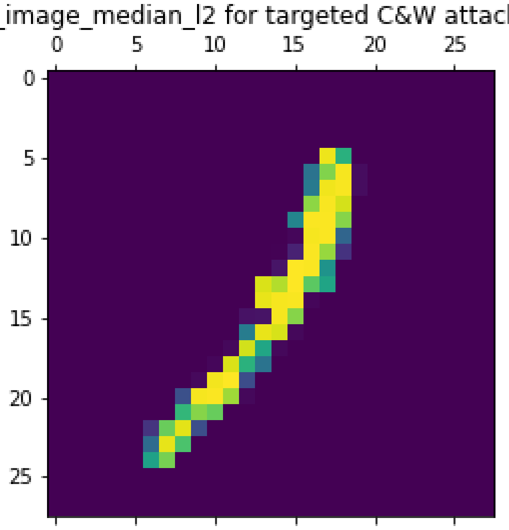


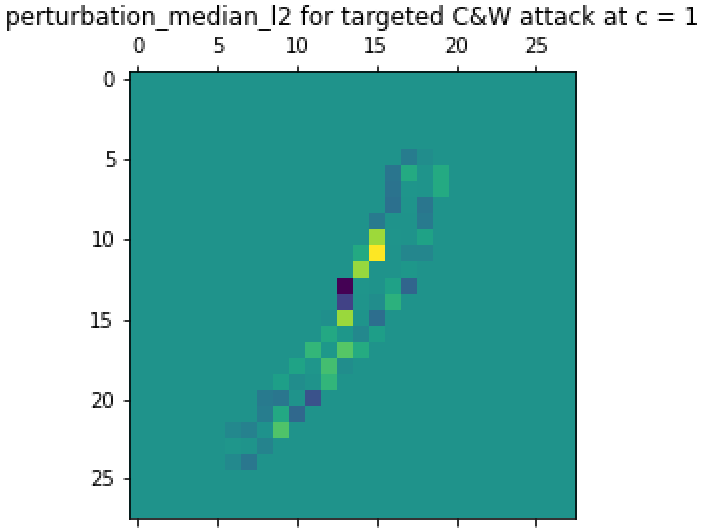
===========MEDIAN L2 ================

original_image_median_l2 for targeted C&W attack at c = 1



attack_image_median_l2 for targeted C&W attack at c = 1



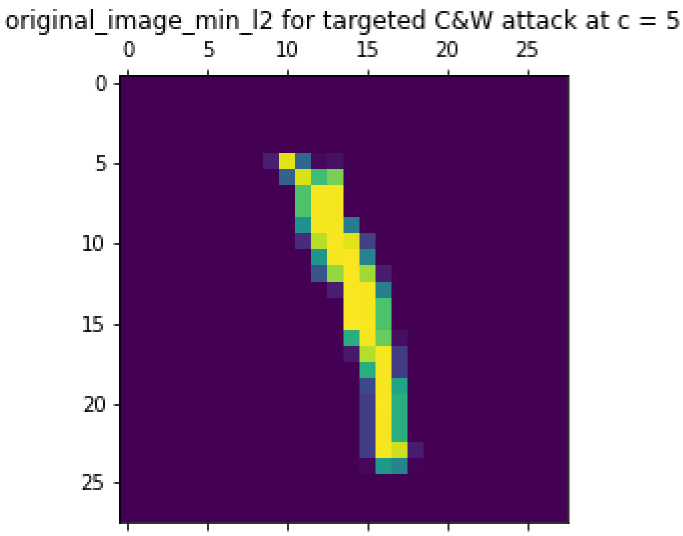


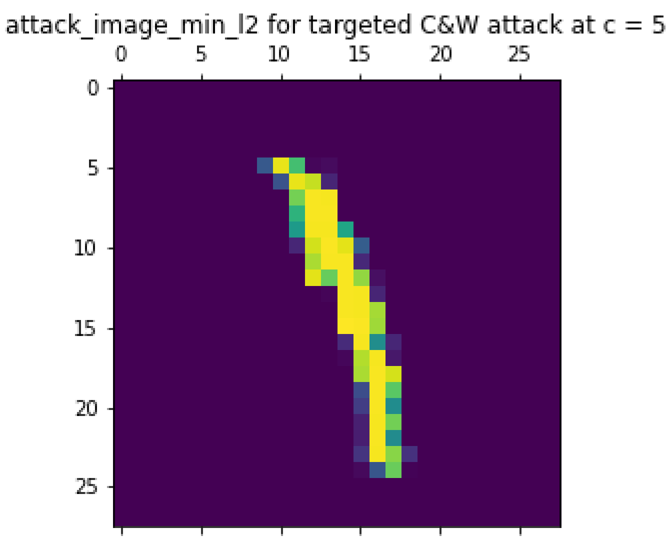
C = 5

In [75]:
```
c_i,c = 2,5
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ===============")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
```

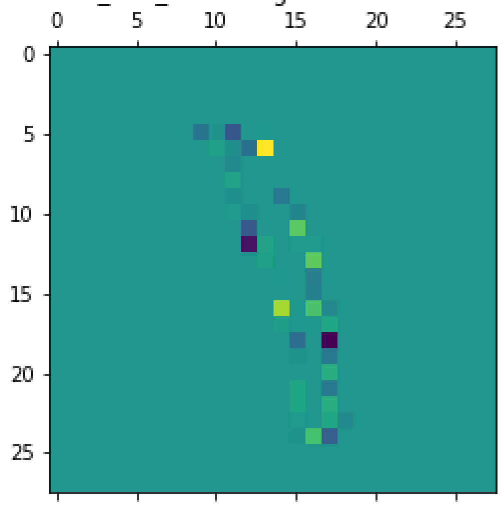
```
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```



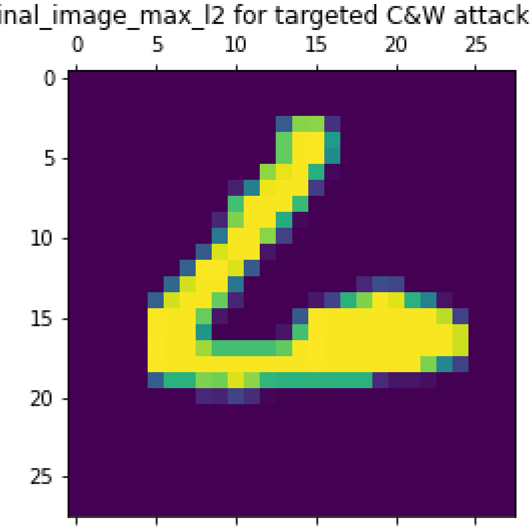


perturbation_min_l2 for targeted C&W attack at c = 5

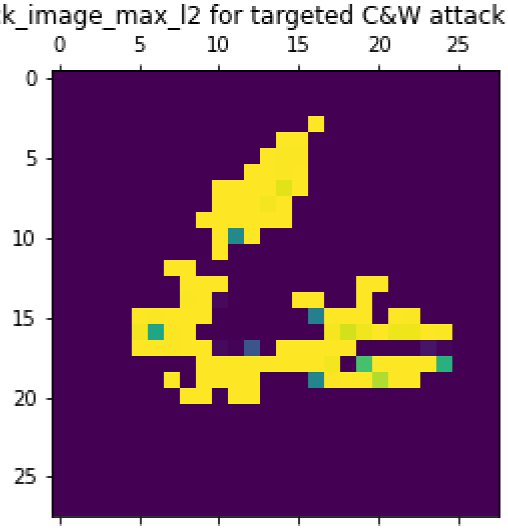


===========MAX L2 ================

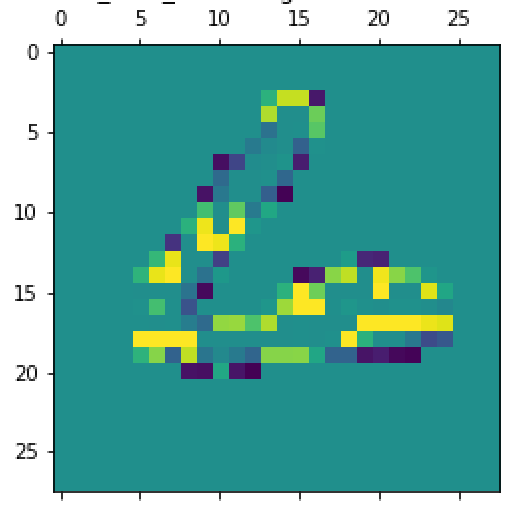
original_image_max_l2 for targeted C&W attack at c = 5



attack_image_max_l2 for targeted C&W attack at c = 5

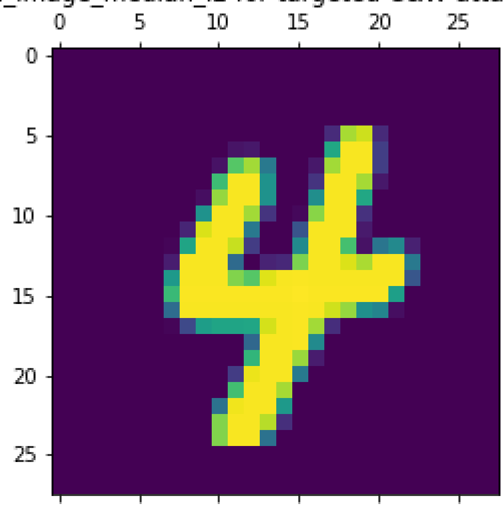


perturbation_max_l2 for targeted C&W attack at c = 5

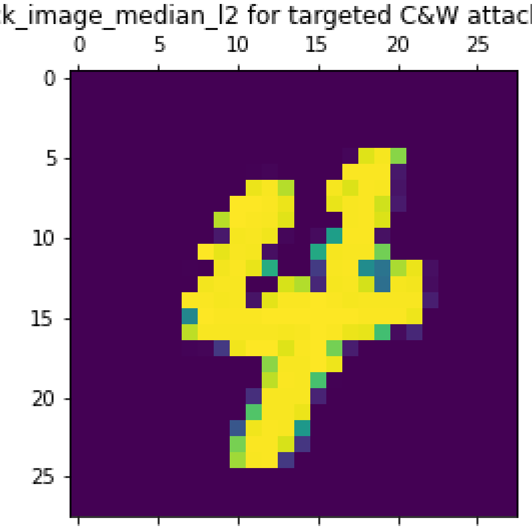


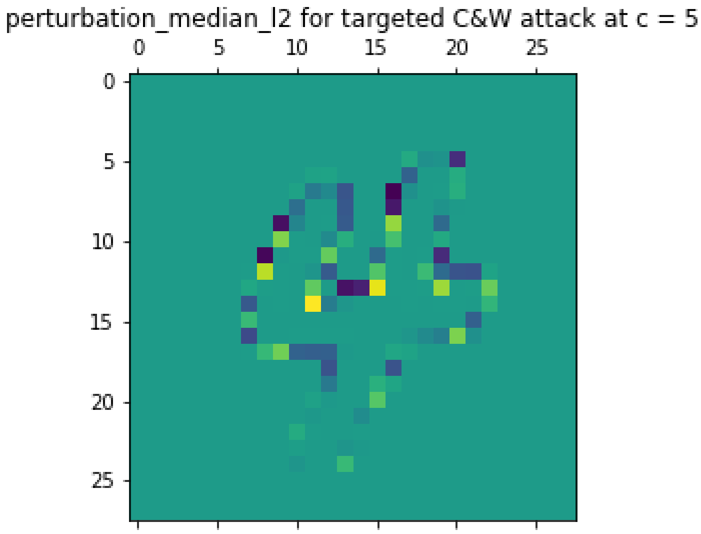

```
===========MEDIAN L2 ================
```

original_image_median_l2 for targeted C&W attack at c = 5



attack_image_median_l2 for targeted C&W attack at c = 5





C = 10

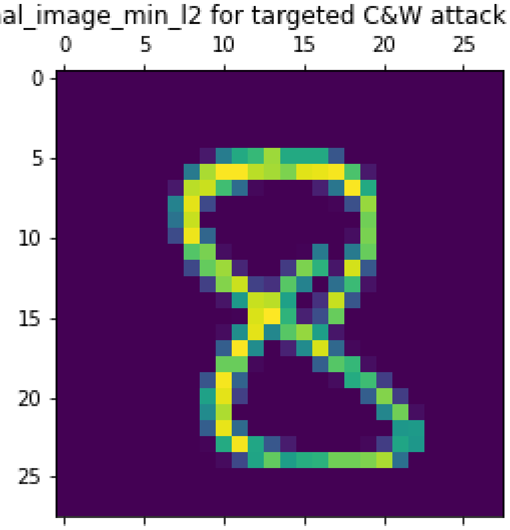
In [76]:

```
c_i,c = 3,10
print(f"===========MIN L2 ================")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
```

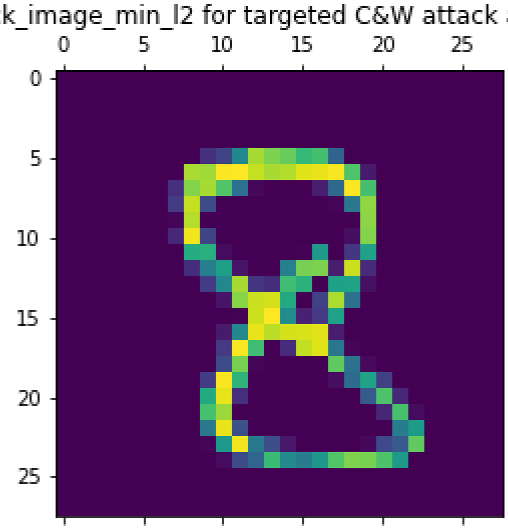
```
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(l2_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

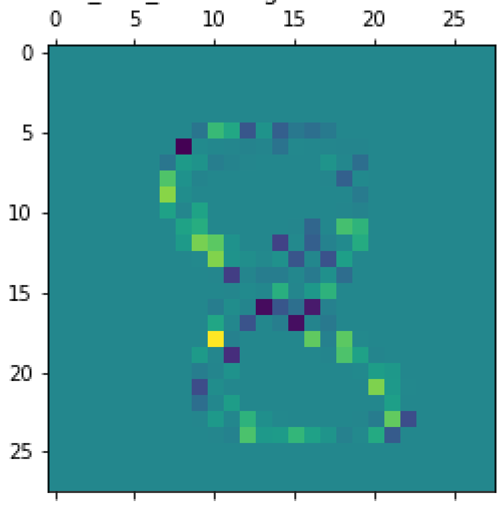
original_image_min_l2 for targeted C&W attack at c = 10



attack_image_min_l2 for targeted C&W attack at c = 10

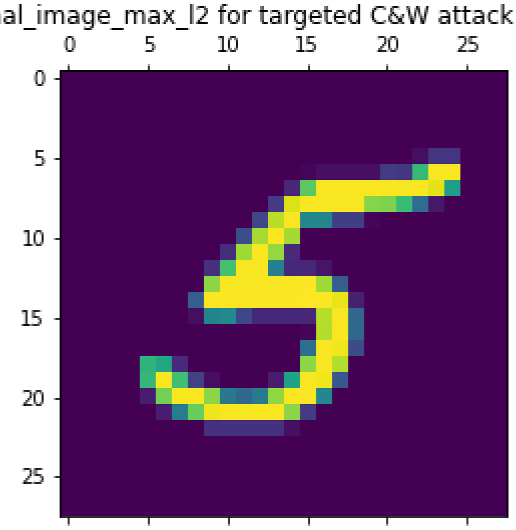


perturbation_min_l2 for targeted C&W attack at c = 10

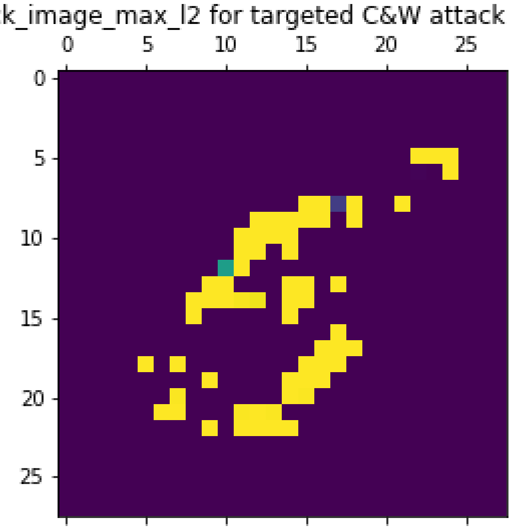



```
===========MAX L2 ================
```

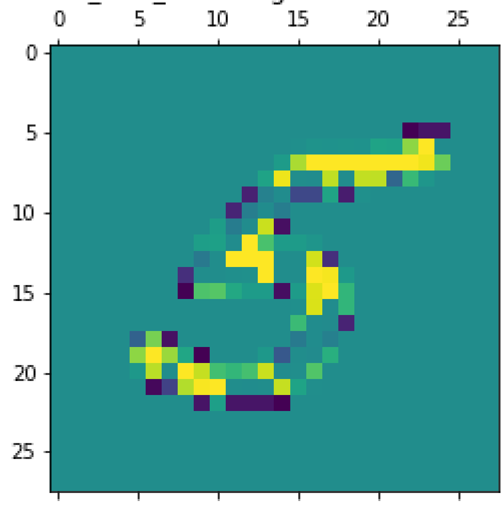
original_image_max_l2 for targeted C&W attack at c = 10



attack_image_max_l2 for targeted C&W attack at c = 10

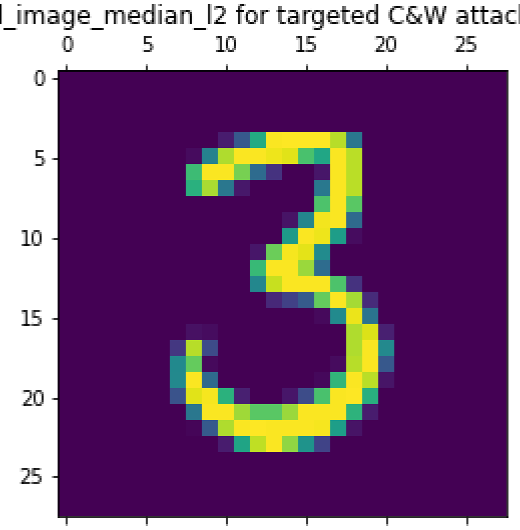


perturbation_max_l2 for targeted C&W attack at c = 10

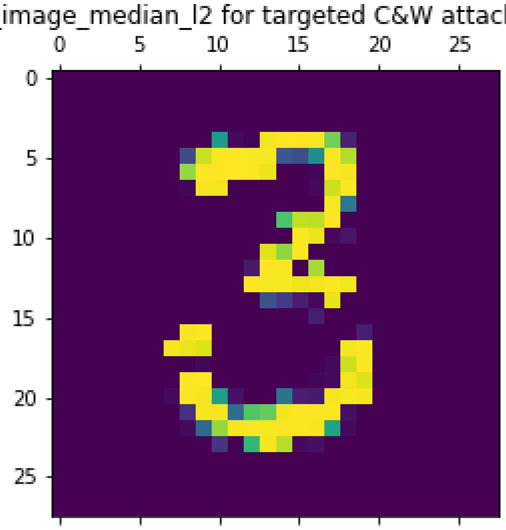


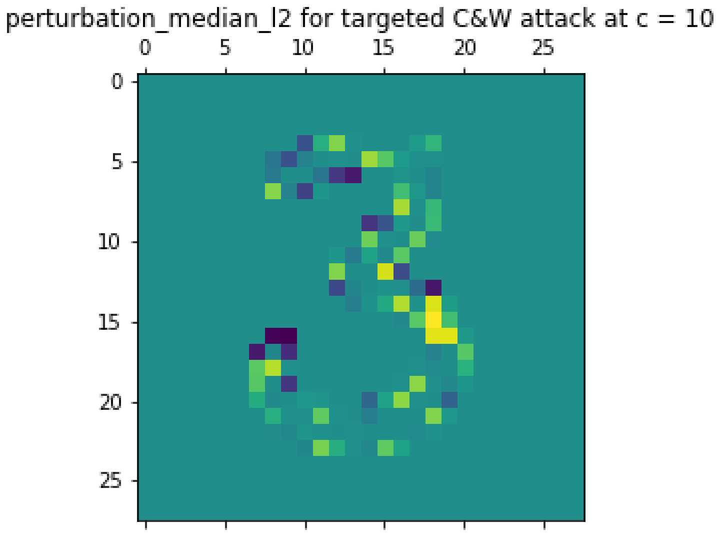
===========MEDIAN L2 ================

original_image_median_l2 for targeted C&W attack at c = 10



attack_image_median_l2 for targeted C&W attack at c = 10





Observations

1. FSGM attacks seem to have much higher perturbation: the difference between original and attacked image is very apparent. Further, the perturbations are not limited to the digit

In [ ]:

In [ ]:

Problem 2

In [1]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import Adam
#initialization code required to make tensorflow work on my systemabs
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
# #disabling eager execution
# tf.compat.v1.disable_eager_execution()
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("Tensorflow version: ",tf.__version__)
import math
from tensorflow.keras.losses import CategoricalCrossentropy
import pandas as pd
from tqdm import trange
import matplotlib.pyplot as plt
```

```
Num GPUs Available:  1
Tensorflow version:  2.4.0
```

load trained model (without softmax) and test images

In [2]:

```
model = keras.models.load_model(r'model')
x_attack_test = np.load(r'x_attack_test.npy')
y_attack_test = np.load(r'y_attack_test.npy')
#make sure it is not trainable
model.trainable = False
```

l2 dist utility

In [3]:

```
#L2 dist is a utility we will use
def l2_squared(a,b):
    #returns the square of the L2 norm of the given matrices (multiple batches)
    #given matrices must have equal dimension
    assert a.shape == b.shape
    return tf.reduce_sum(tf.math.square(a - b),axis=[1,2,3])
```

General C&W attack framework

In [7]:

```
class CustomCW:
    ## my custom custom CW attack class: customizable objective function
    def __init__(self,c,objective,num_iters,learning_rate,verbose=False):
        self.c = c
        self.objective = objective
        self.num_iters = num_iters
        self.lr=learning_rate
        self.verbose = verbose
```

```
    def delta(self,w):
        #simply applies tanh to the given input multiplies by half and adds 1 and adds
        #this is the perturbation
        out =  0.5*(tf.tanh(w)+1)
        return out


    def get_loss(self,x,adv_x):
        #x and transformed to tanh space for optimization as per C&W code: this improve
        loss = l2_squared(adv_x,self.delta(x)) + self.c*self.objective.get_obj(adv_x)
        #sum up losses across images
        return tf.reduce_sum(loss)



    def attack(self,images):
        ##attack the images so they  are misclassified to the target class

        #w and orginal images as tensorflow variables
        w = tf.Variable(np.zeros(images.shape,dtype=np.float32))
        original_images = tf.Variable(images,dtype=np.float32)

        optimizer = Adam(self.lr)

        #so we can stop training wehen loss stops increasing
        prev_loss = math.inf


        for it in trange(self.num_iters):
            with tf.GradientTape() as tape:
                delta = self.delta(w)
                adversarial_images = original_images + delta

                loss = self.get_loss(original_images,adversarial_images)

                #stop training if loss stops decreasing by 0.1
                if(abs(prev_loss-loss)<0.1):
                    break


            #update w using gradients
            gradients = tape.gradient(loss, [w])
            optimizer.apply_gradients(zip(gradients, [w]))

            #printing out progress
            if(it%100==0 and self.verbose):
                print(f"iteration {it}: loss = {loss}")


        #return images
        return adversarial_images
```

(1) Szedegy et al objective

In [8]:

```
class szedegy:
    #thi class is a function object that applies the loss function from szedegy et al:
    #the objective in question here is simply the cross entropy loss on the target clas

    def __init__(self, model, target, num_classes):
        #init with model we are attacking and target

        self.model = model
        self.num_classes = num_classes

        #transform target to categorical space
        target_cat = [0]*num_classes
        target_cat[target] = 1
        self.target = tf.constant([target_cat])

        #loss object: we use categorical cross entropy
        self.loss = CategoricalCrossentropy(from_logits=True)

    def get_obj(self,adv_x):
        batch_size = len(adv_x)
        #get logits on adv_x
        logits = self.model(adv_x)
        #simply returns cross entropy loss wrt to target

        return self.loss(tf.broadcast_to(self.target,[batch_size,self.num_classes]),log
```

(2) f_6 objective from C&W paper

In [9]:

```
class f6:

    #this class is a function object that applies f6 loss from the C&W Paper
    def __init__(self,model,target,num_classes):
        self.model = model
        self.target = target
        self.num_classes = num_classes

    def get_max_z_not_t(self,logits):
        #gets the maximum logit thats not the target class
        return tf.reduce_max(tf.gather(logits,indices = [i for i in list(range(self.num_
    def get_obj(self,x_adv):
        #returns the f6 objective of the given adversarial image
        logits = self.model(x_adv)
        return tf.math.maximum(self.get_max_z_not_t(logits)-logits[:,self.target],tf.co
```

Problem 2a - adversarial success as a function of c

In [13]:
```
#same learning rate as ART to standardize
learning_rate = 0.01
num_iters = 50
constants = constants = [0.5,1,5,10]

target = 7
num_test_samples = len(x_attack_test)
num_classes = 10
```

Szedegy objective

In [14]:
```
#store attack metrics for various c's in a list of dictionaries
szedegy_attack_metrics = []
for idx,c in enumerate(constants):
    ## Szedegy objective

    print(f"=====attacking at c = {c}=====")
    #generate attack images
    szedegy_obj = szedegy(model,target,num_classes)
    sz_cw = CustomCW(c,szedegy_obj,num_iters,learning_rate)
    sz_attacks = sz_cw.attack(x_attack_test)

    #predictions
    sz_predict = model.predict(sz_attacks)

    #calculate success rate
    success_mask = np.argmax(sz_predict, axis=1) == target
    attack_success_rate = sum(success_mask)/num_test_samples


    #calculate l2 norm
    l2dist = np.sqrt(np.sum(np.square(x_attack_test - sz_attacks).reshape(num_test_samp

    #create series out of l2 dist to pull out min,max,median
    l2dist = pd.Series(l2dist,index=range(num_test_samples))


    #Report attack success rate and avg l2 norm
    print(f"Attack Success Rate targeted C&W attack with Szedegy obj at c= {c} : {round
    print(f"Avg L2 Norm on targeted C&W with Szedegy obj at c = {c} : {round(l2dist.mea



    min_l2_idx = pd.Series.idxmin(l2dist[success_mask])
    max_l2_idx = pd.Series.idxmax(l2dist[success_mask])
    median_l2_idx = int(l2dist[success_mask].sort_values(ignore_index=False).reset_inde


    #append the l2 metrics
    szedegy_attack_metrics.append(
    {
        'c':c,
        'attack_success_rate':attack_success_rate,
        'min_l2_idx':min_l2_idx,
```

```
        'max_l2_idx':max_l2_idx,
        'median_l2_idx':median_l2_idx,

        'min_l2':l2dist[min_l2_idx],
        'max_l2':l2dist[max_l2_idx],
        'median_l2':l2dist[median_l2_idx],

        'original_image_min_l2':x_attack_test[min_l2_idx],
        'original_image_max_l2':x_attack_test[max_l2_idx],
        'original_image_median_l2':x_attack_test[median_l2_idx],

        'attack_image_min_l2':sz_attacks[min_l2_idx],
        'attack_image_max_l2':sz_attacks[max_l2_idx],
        'attack_image_median_l2':sz_attacks[median_l2_idx],

        'perturbation_min_l2':x_attack_test[min_l2_idx]- sz_attacks[min_l2_idx],
        'perturbation_max_l2':x_attack_test[max_l2_idx] - sz_attacks[max_l2_idx],
        'perturbation_median_l2':x_attack_test[median_l2_idx] - sz_attacks[median_l2_id




    })
```

```
=====attacking at c = 0.5=====
100%|█████████████████████████████████████████
████| 50/50 [00:00<00:00, 63.37it/s]
Attack Success Rate targeted C&W attack with Szedegy obj at c= 0.5 : 35.0%
Avg L2 Norm on targeted C&W with Szedegy obj at c = 0.5 : 13.279999732971191
=====attacking at c = 1=====
100%|█████████████████████████████████████████
████| 50/50 [00:00<00:00, 62.74it/s]
Attack Success Rate targeted C&W attack with Szedegy obj at c= 1 : 56.0%
Avg L2 Norm on targeted C&W with Szedegy obj at c = 1 : 13.350000381469727
=====attacking at c = 5=====
100%|█████████████████████████████████████████
████| 50/50 [00:00<00:00, 63.69it/s]
Attack Success Rate targeted C&W attack with Szedegy obj at c= 5 : 71.0%
Avg L2 Norm on targeted C&W with Szedegy obj at c = 5 : 13.649999618530273
=====attacking at c = 10=====
100%|█████████████████████████████████████████
████| 50/50 [00:00<00:00, 61.73it/s]
Attack Success Rate targeted C&W attack with Szedegy obj at c= 10 : 71.0%
Avg L2 Norm on targeted C&W with Szedegy obj at c = 10 : 13.800000190734863
```

In [28]:

```
szedegy_df = pd.DataFrame(szedegy_attack_metrics)
```

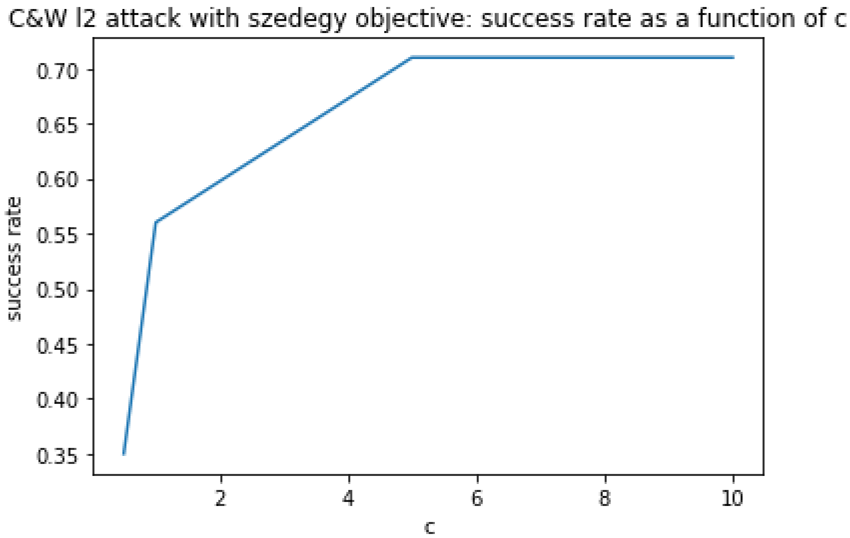
In [29]:

```
fig = plt.figure()
ax = plt.axes()
ax.plot(szedegy_df['c'], szedegy_df['attack_success_rate'])
plt.title("C&W l2 attack with szedegy objective: success rate as a function of c")
```

```
plt.xlabel("c")
plt.ylabel("success rate")
```

Out[29]:

```
Text(0, 0.5, 'success rate')
```



f_6 objective from C&W paper

In [15]:

```
#store attack metrics for various c's in a list of dictionaries
f6_attack_metrics = []
for idx,c in enumerate(constants):
    ## f6 objective

    print(f"=====attacking at c = {c}=====")
    #generate attack images
    f6_obj = f6(model,target,num_classes)
    #higher learning rates for lower c's
    f6_cw = CustomCW(c,f6_obj,num_iters,learning_rate)
    f6_attacks = f6_cw.attack(x_attack_test)

    #predictions
    f6_predict = model.predict(f6_attacks)

    #calculate success rate
    success_mask = np.argmax(f6_predict, axis=1) == target
    attack_success_rate = sum(success_mask)/num_test_samples


    #calculate l2 norm
    l2dist = np.sqrt(np.sum(np.square(x_attack_test - f6_attacks).reshape(num_test_samp
    #create series out of l2 dist to pull out min,max,median
    l2dist = pd.Series(l2dist,index=range(num_test_samples))


    #Report attack success rate and avg l2 norm
    print(f"Attack Success Rate targeted C&W attack with f6 obj at c= {c} : {round(atta
    print(f"Avg L2 Norm on targeted C&W with f6 obj at c = {c} : {round(l2dist.mean(),2

    min_l2_idx = pd.Series.idxmin(l2dist[success_mask])
    max_l2_idx = pd.Series.idxmax(l2dist[success_mask])
    median_l2_idx = int(l2dist[success_mask].sort_values(ignore_index=False).reset_inde
```

```
    #append the l2 metrics
    f6_attack_metrics.append(
    {
        'c':c,
        'attack_success_rate':attack_success_rate,
        'min_l2_idx':min_l2_idx,
        'max_l2_idx':max_l2_idx,
        'median_l2_idx':median_l2_idx,

        'min_l2':l2dist[min_l2_idx],
        'max_l2':l2dist[max_l2_idx],
        'median_l2':l2dist[median_l2_idx],

        'original_image_min_l2':x_attack_test[min_l2_idx],
        'original_image_max_l2':x_attack_test[max_l2_idx],
        'original_image_median_l2':x_attack_test[median_l2_idx],

        'attack_image_min_l2':f6_attacks[min_l2_idx],
        'attack_image_max_l2':f6_attacks[max_l2_idx],
        'attack_image_median_l2':f6_attacks[median_l2_idx],

        'perturbation_min_l2':x_attack_test[min_l2_idx]- f6_attacks[min_l2_idx],
        'perturbation_max_l2':x_attack_test[max_l2_idx] - f6_attacks[max_l2_idx],
        'perturbation_median_l2':x_attack_test[median_l2_idx] - f6_attacks[median_l2_id




    })
```

```
=====attacking at c = 0.5=====
100%|██████████████████████████████████████████
████| 50/50 [00:00<00:00, 59.17it/s]
Attack Success Rate targeted C&W attack with f6 obj at c= 0.5 : 32.0%
Avg L2 Norm on targeted C&W with f6 obj at c = 0.5 : 13.289999961853027
=====attacking at c = 1=====
100%|██████████████████████████████████████████
████| 50/50 [00:00<00:00, 58.89it/s]
Attack Success Rate targeted C&W attack with f6 obj at c= 1 : 48.0%
Avg L2 Norm on targeted C&W with f6 obj at c = 1 : 13.359999656677246
=====attacking at c = 5=====
100%|██████████████████████████████████████████
████| 50/50 [00:00<00:00, 59.74it/s]
Attack Success Rate targeted C&W attack with f6 obj at c= 5 : 73.0%
Avg L2 Norm on targeted C&W with f6 obj at c = 5 : 13.680000305175781
=====attacking at c = 10=====
100%|██████████████████████████████████████████
████| 50/50 [00:00<00:00, 59.17it/s]
Attack Success Rate targeted C&W attack with f6 obj at c= 10 : 75.0%
Avg L2 Norm on targeted C&W with f6 obj at c = 10 : 13.819999694824219
```

In [30]:

```
f6_df = pd.DataFrame(f6_attack_metrics)
```

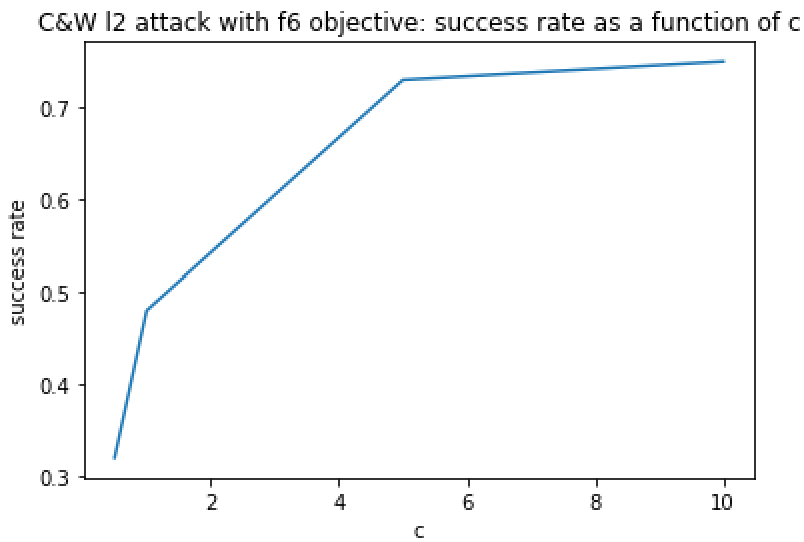
In [31]:

```
fig = plt.figure()
ax = plt.axes()
ax.plot(f6_df['c'], f6_df['attack_success_rate'])
plt.title("C&W l2 attack with f6 objective: success rate as a function of c")
```

```
plt.xlabel("c")
plt.ylabel("success rate")
```

Out[31]:

```
Text(0, 0.5, 'success rate')
```



Observations

1. Szegedy objective seems to perform slightly worse than the f_6 objective, but only for the high values of c like 5 and 10. It performs better than f_6 for low values of c
2. Both attacks have very similar avg l2 perturbation norms, which dont vary much as per c.

Problem 2b Min,Max and Median l2 distance images for various values of c: Successful Carlini Wagner attacks

Szedegy objective

In [34]:

```
szedegy_df[['c','attack_success_rate','min_l2_idx','min_l2','max_l2_idx','max_l2','medi
```

Out[34]:

	c	attack_success_rate	min_l2_idx	min_l2	max_l2_idx	max_l2	median_l2_idx	median_l2
0	0.5	0.35	10	12.843558	50	13.571725	1	13.209056
1	1.0	0.56	51	12.629746	55	13.789464	53	13.385027
2	5.0	0.71	77	13.042408	28	14.113783	82	13.674903
3	10.0	0.71	1	13.203969	28	14.229536	82	13.847279

C = 0.5

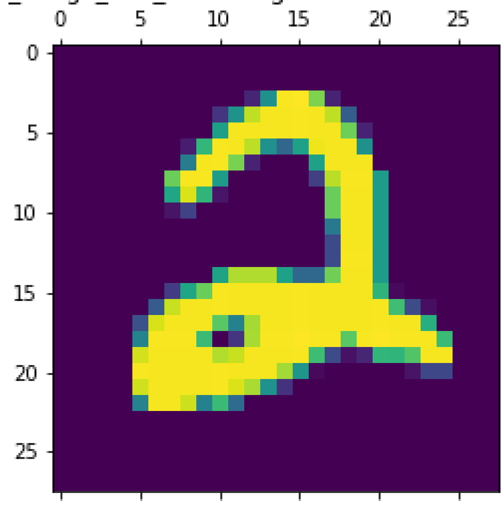
In [16]:

```
c_i,c = 0,0.01
print(f"==========MIN L2 ================")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
```

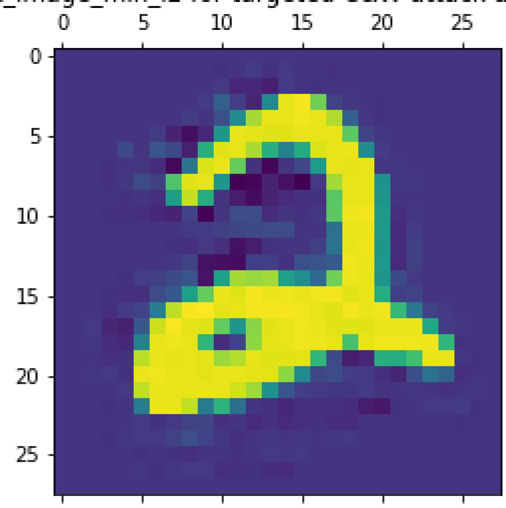
```
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

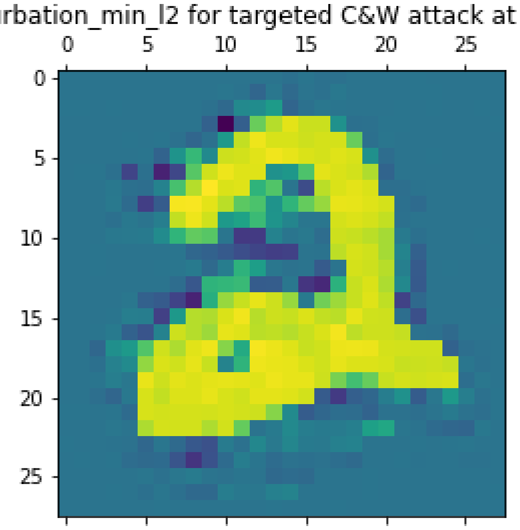
original_image_min_l2 for targeted C&W attack at c = 0.01



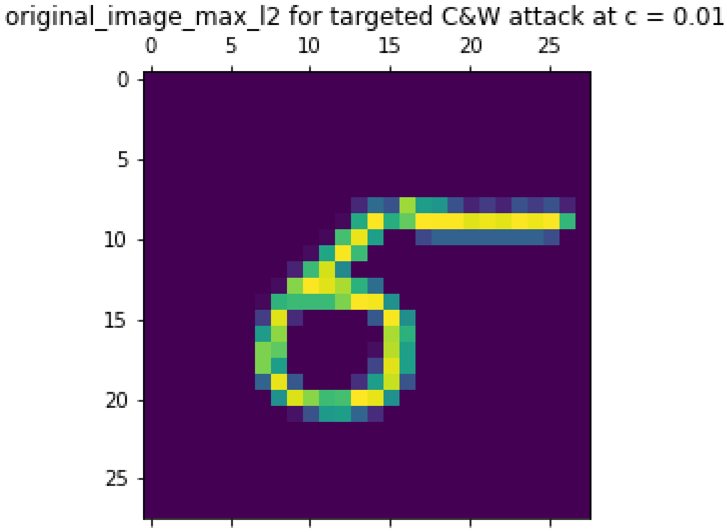
attack_image_min_l2 for targeted C&W attack at c = 0.01

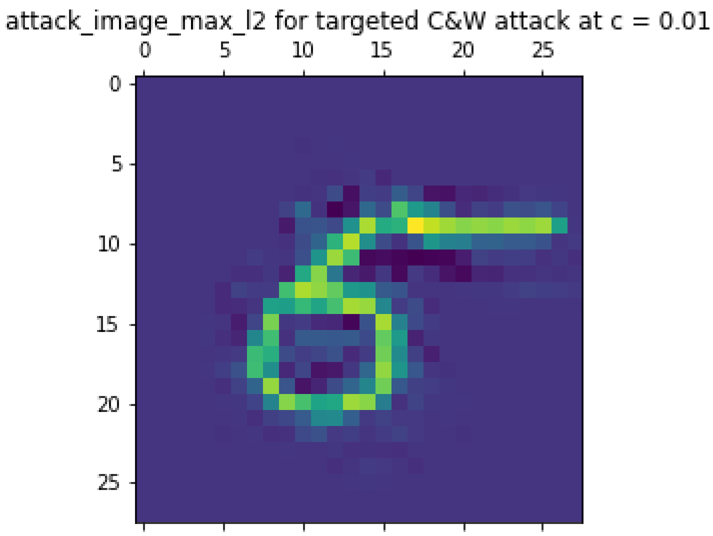


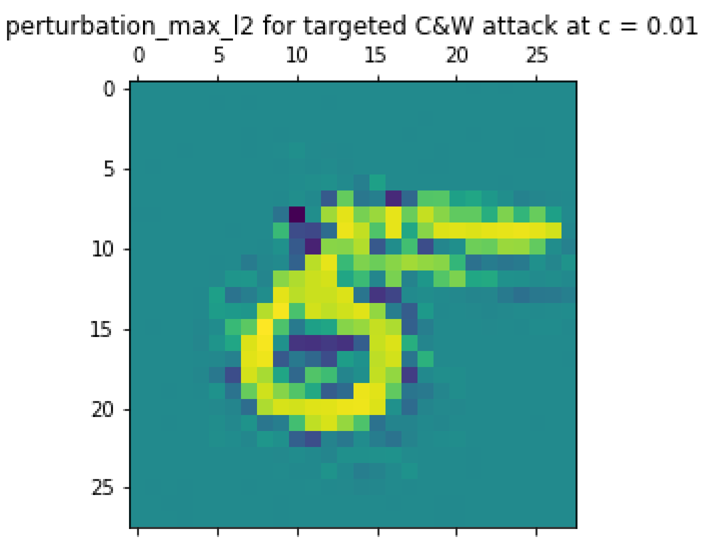
perturbation_min_l2 for targeted C&W attack at c = 0.01



```
===========MAX L2 ================
```

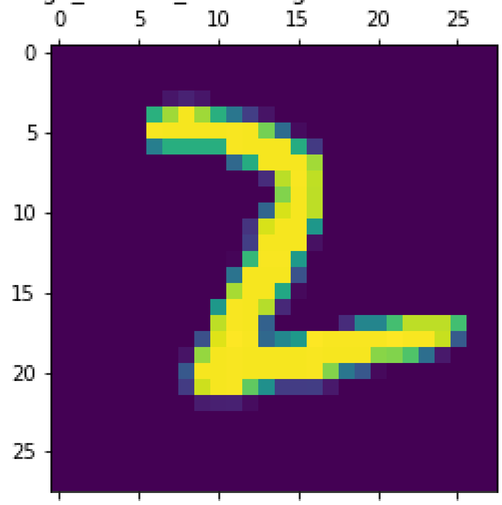

original_image_max_l2 for targeted C&W attack at c = 0.01
0 5 10 15 20 25
0 5 10 15 20 25


attack_image_max_l2 for targeted C&W attack at c = 0.01
0 5 10 15 20 25
0 5 10 15 20 25

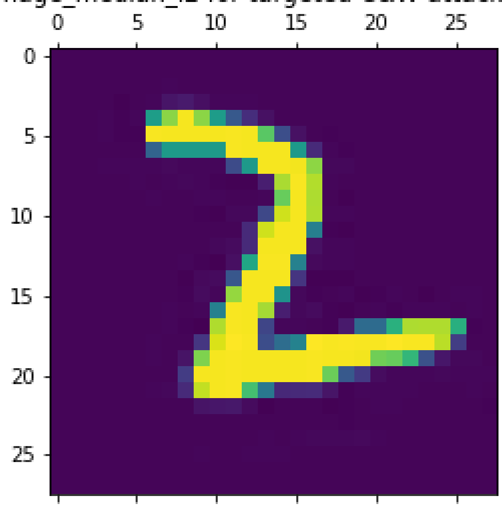

perturbation_max_l2 for targeted C&W attack at c = 0.01
0 5 10 15 20 25
0 5 10 15 20 25

```
===========MEDIAN L2 ================
```

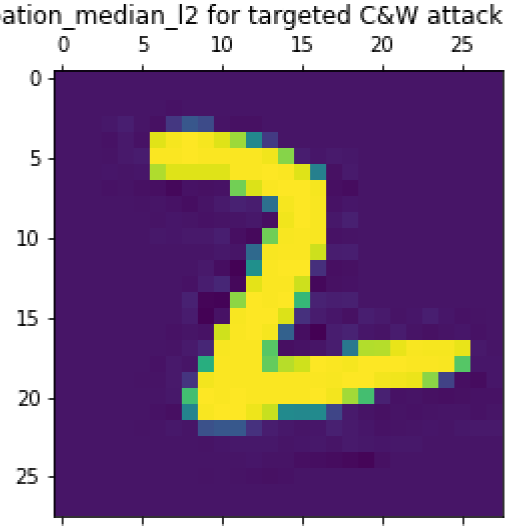
original_image_median_l2 for targeted C&W attack at c = 0.01



attack_image_median_l2 for targeted C&W attack at c = 0.01



perturbation_median_l2 for targeted C&W attack at c = 0.01



C = 1

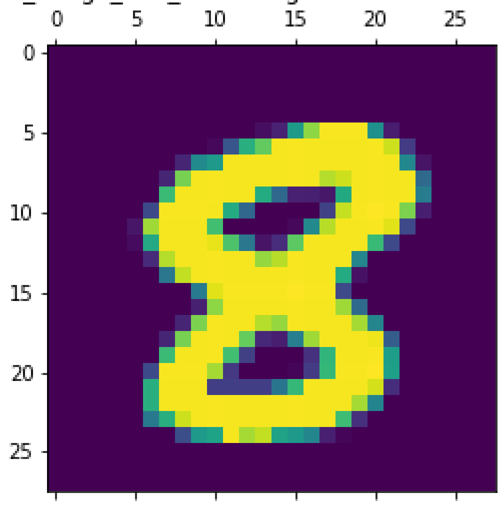
In [18]:

```
c_i,c = 1,1
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
```

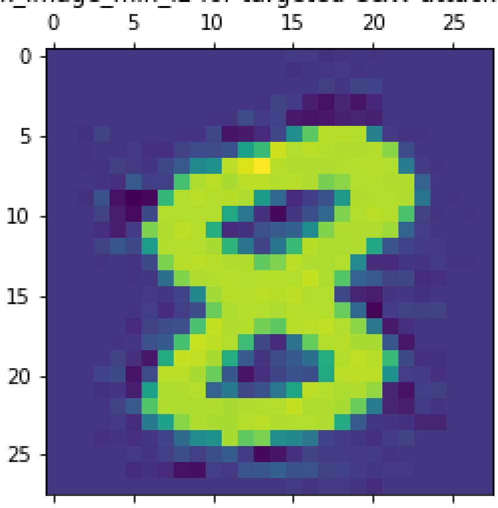
```
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

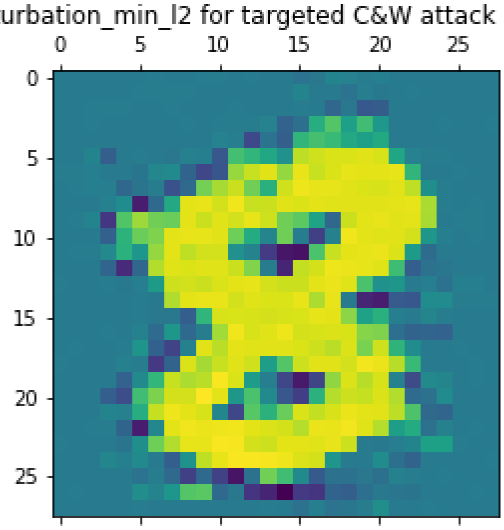
original_image_min_l2 for targeted C&W attack at c = 1



attack_image_min_l2 for targeted C&W attack at c = 1

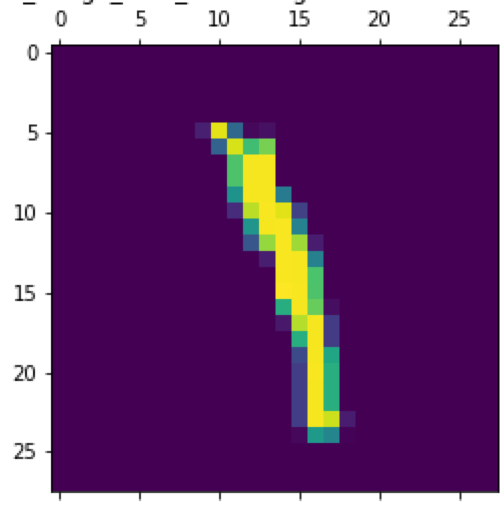


perturbation_min_l2 for targeted C&W attack at c = 1

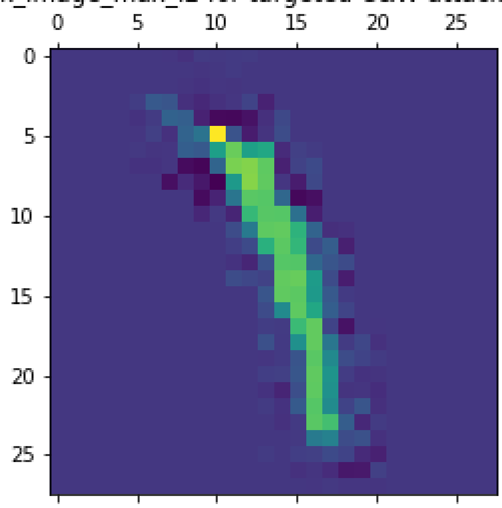


===========MAX L2 ================

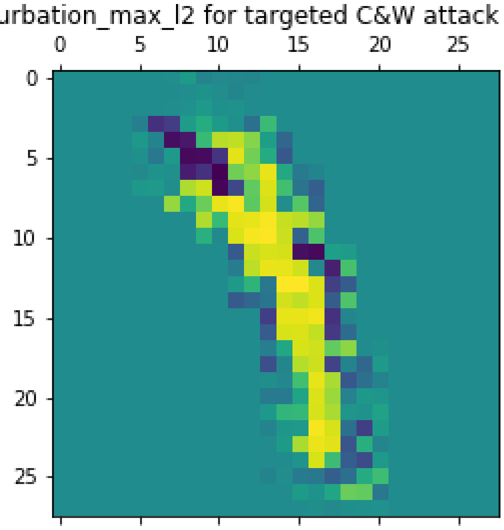
original_image_max_l2 for targeted C&W attack at c = 1



attack_image_max_l2 for targeted C&W attack at c = 1

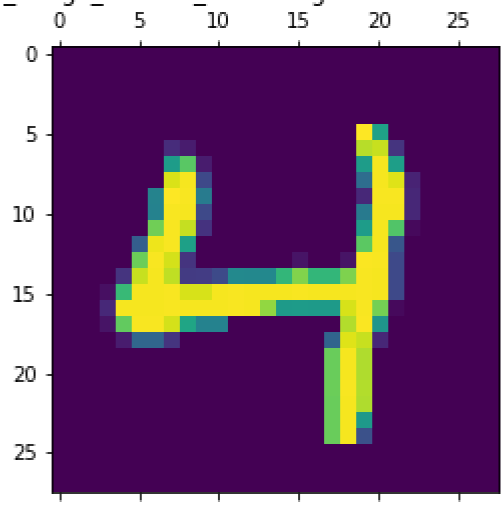


perturbation_max_l2 for targeted C&W attack at c = 1

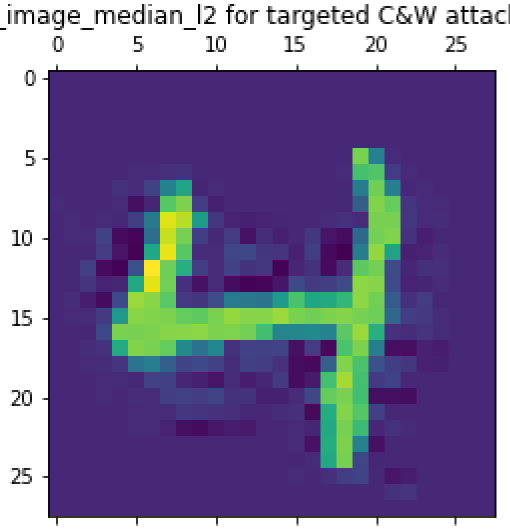



```
===========MEDIAN L2 ================
```

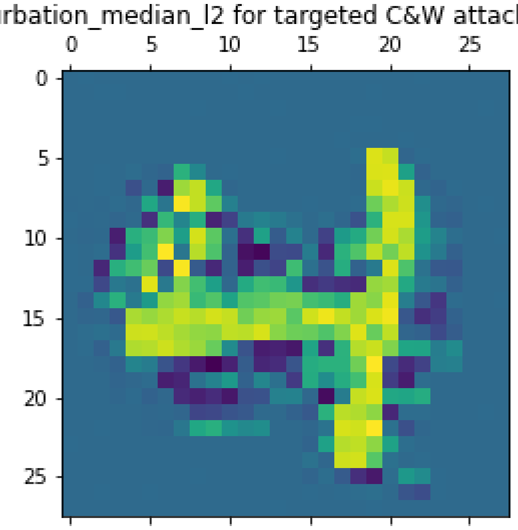
original_image_median_l2 for targeted C&W attack at c = 1



attack_image_median_l2 for targeted C&W attack at c = 1



perturbation_median_l2 for targeted C&W attack at c = 1



C= 5

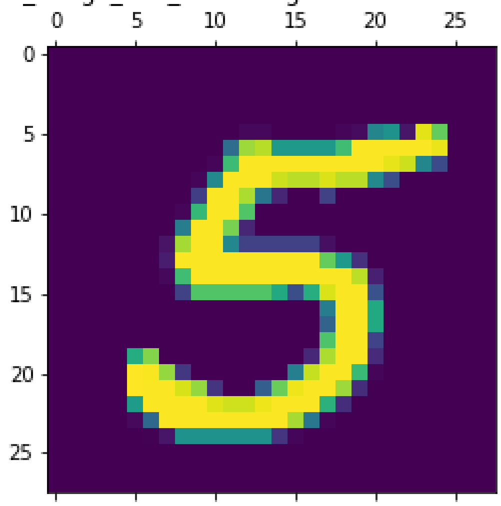
In [19]:

```
c_i,c = 2,5
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
```

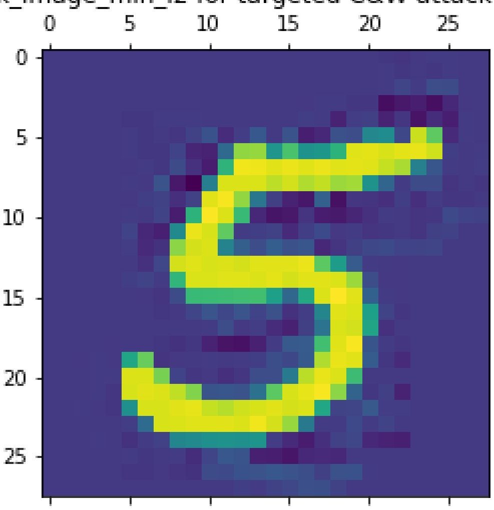
```
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

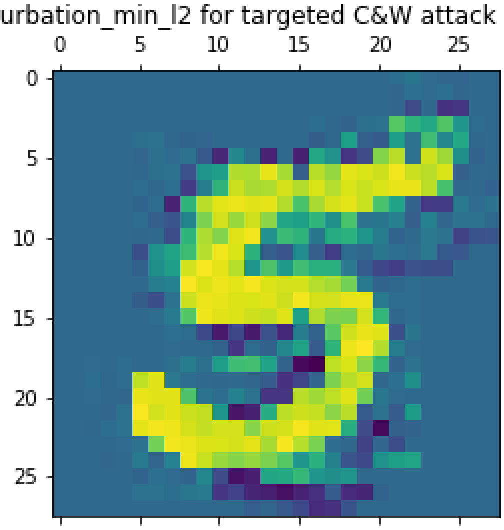
original_image_min_l2 for targeted C&W attack at c = 5



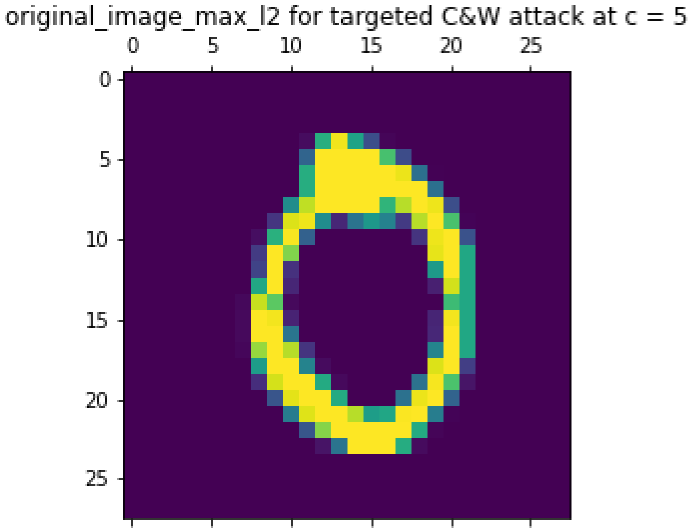
attack_image_min_l2 for targeted C&W attack at c = 5

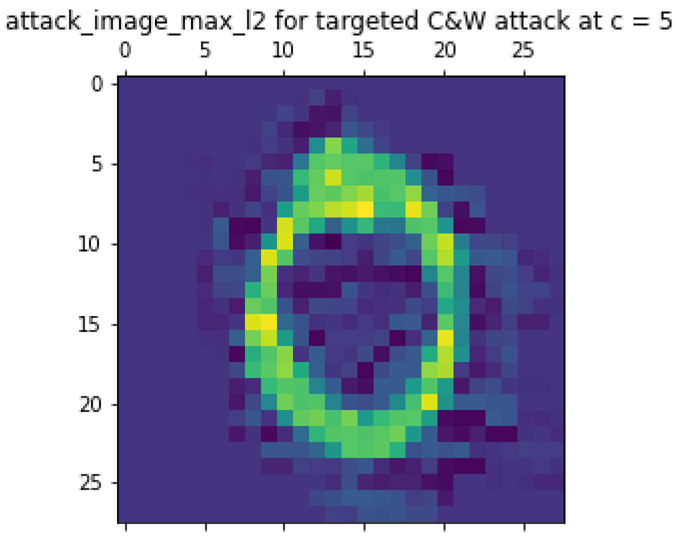


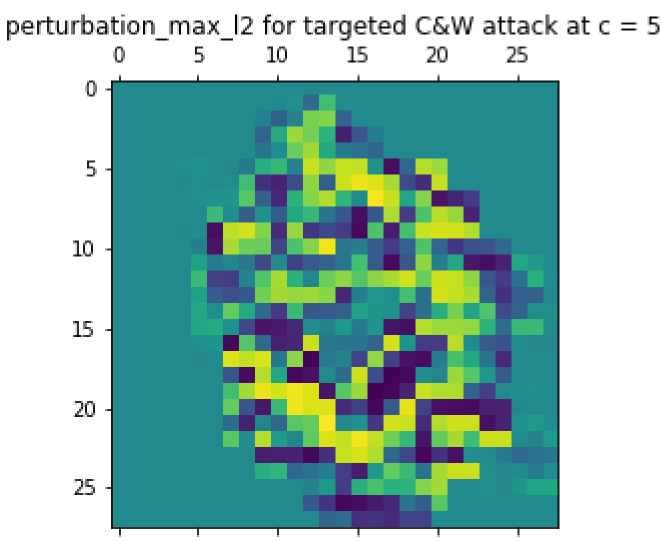
perturbation_min_l2 for targeted C&W attack at c = 5



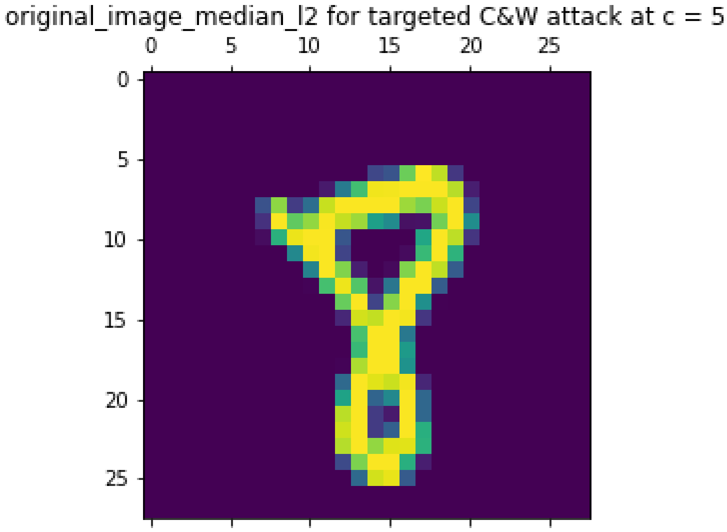
```
===========MAX L2 ================
```

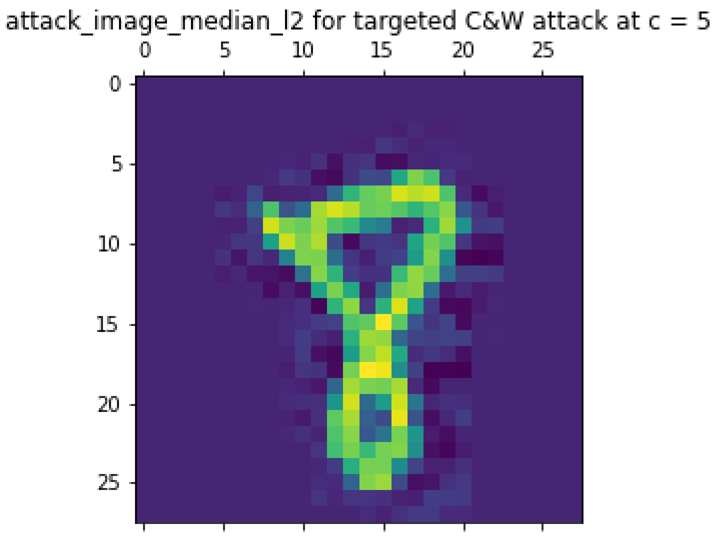

original_image_max_l2 for targeted C&W attack at c = 5
0
5
10
15
20
25

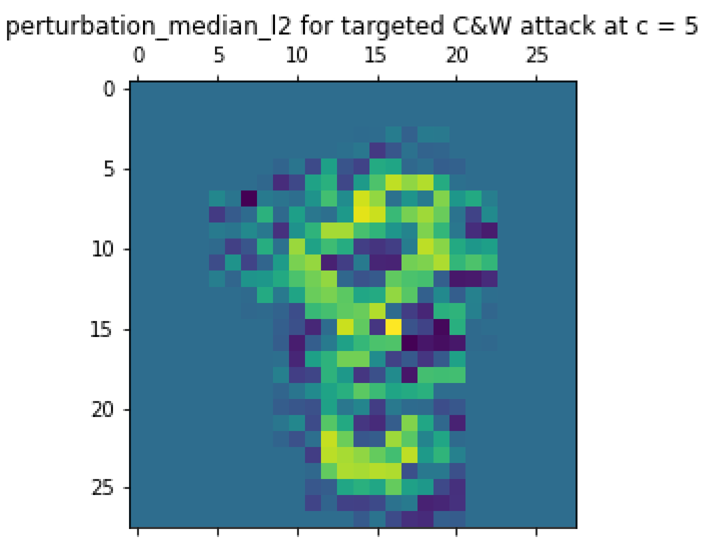

attack_image_max_l2 for targeted C&W attack at c = 5
0
5
10
15
20
25


perturbation_max_l2 for targeted C&W attack at c = 5
0
5
10
15
20
25

```
===========MEDIAN L2 ================
```







C = 10

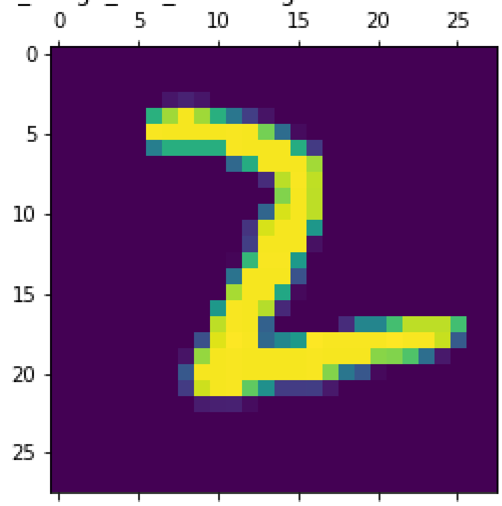
In [20]:

```
c_i,c = 3,10
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
```

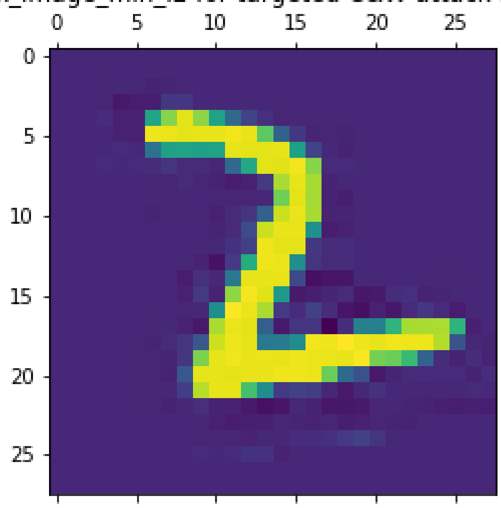
```
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(szedegy_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

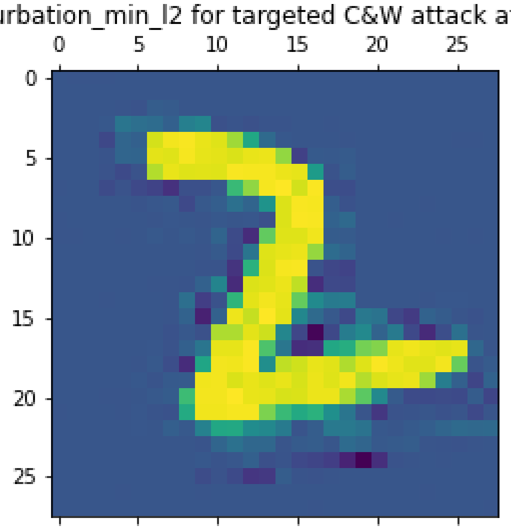
original_image_min_l2 for targeted C&W attack at c = 10



attack_image_min_l2 for targeted C&W attack at c = 10

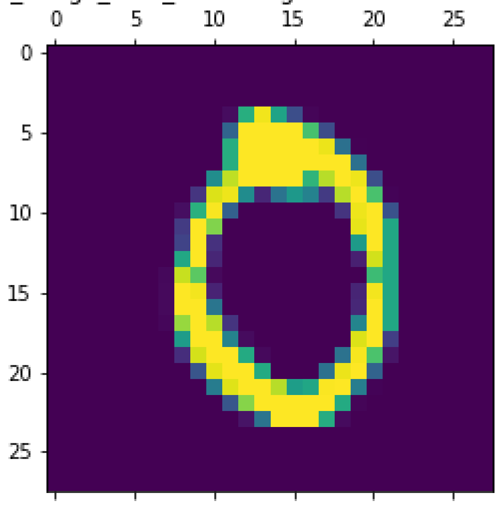


perturbation_min_l2 for targeted C&W attack at c = 10

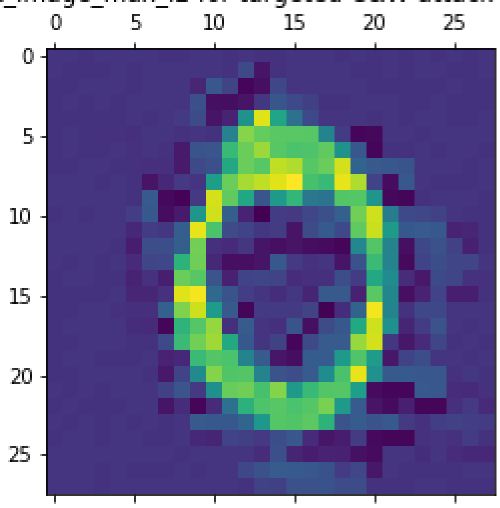


```
===========MAX L2 ================
```

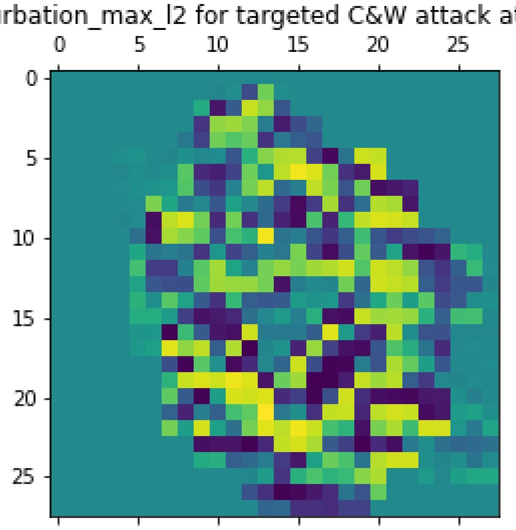
original_image_max_l2 for targeted C&W attack at c = 10



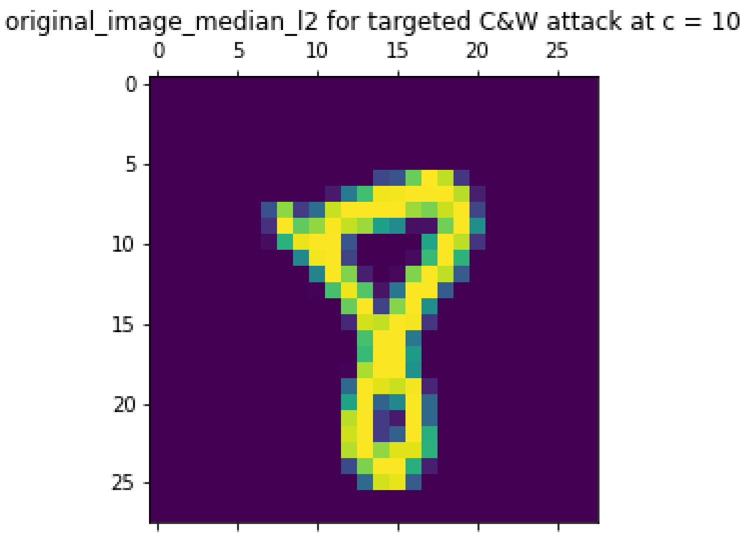
attack_image_max_l2 for targeted C&W attack at c = 10

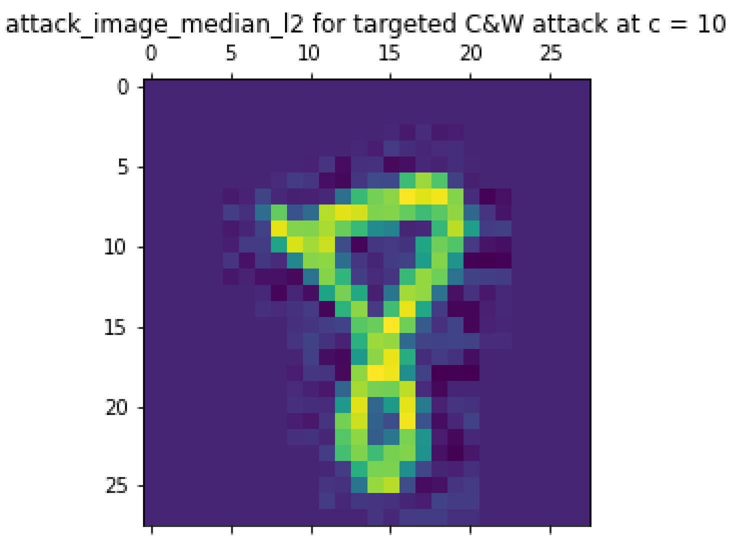


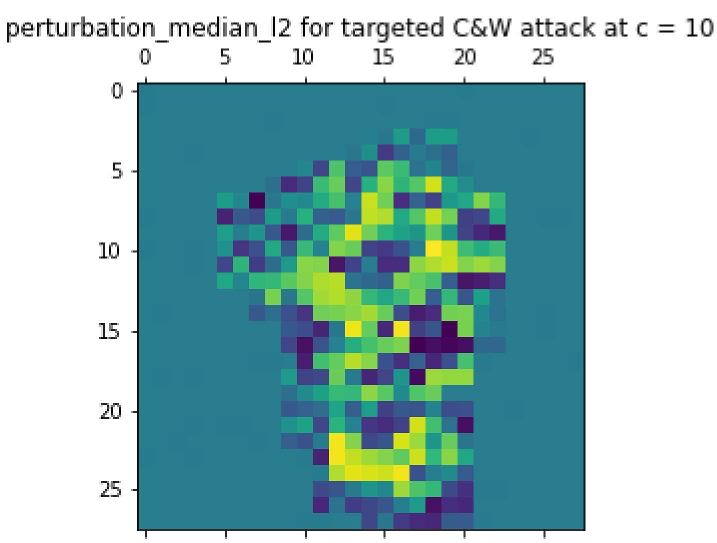
perturbation_max_l2 for targeted C&W attack at c = 10




```
===========MEDIAN L2 ================
```







f6 Objective

In [35]:
```
f6_df[['c','attack_success_rate','min_l2_idx','min_l2','max_l2_idx','max_l2','median_l2
```

Out[35]:

	c	attack_success_rate	min_l2_idx	min_l2	max_l2_idx	max_l2	median_l2_idx	median_l2

	c	attack_success_rate	min_l2_idx	min_l2	max_l2_idx	max_l2	median_l2_idx	median_l2
0	0.5	0.32	8	12.868443	60	13.604152	65	13.360970
1	1.0	0.48	51	12.648972	55	13.793358	53	13.389554
2	5.0	0.73	75	13.034954	28	14.100512	14	13.693420
3	10.0	0.75	75	13.034954	28	14.231073	9	13.847732

C = 0.5

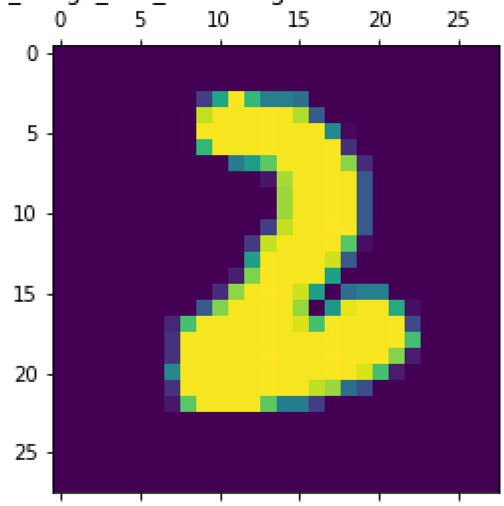
In [22]:

```
c_i,c = 0,0.01
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ===============")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MEDIAN L2 ===============")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

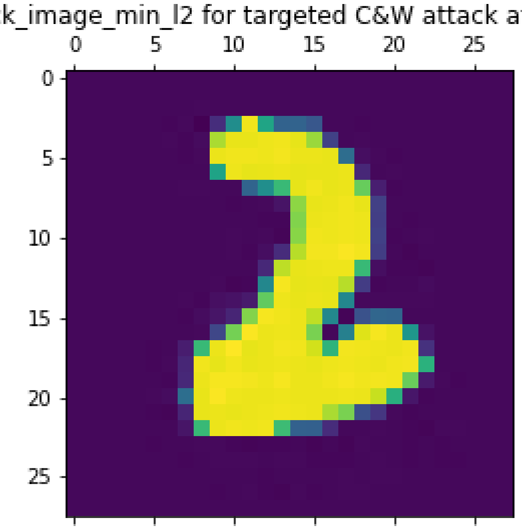
```
    viz_type = "attack_image_"
    viz_key = viz_type + l2_type
    plt.matshow(f6_attack_metrics[c_i][viz_key])
    plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
    plt.show()

    viz_type = "perturbation_"
    viz_key = viz_type + l2_type
    plt.matshow(f6_attack_metrics[c_i][viz_key])
    plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
    plt.show()
```

```
===========MIN L2 ================
```

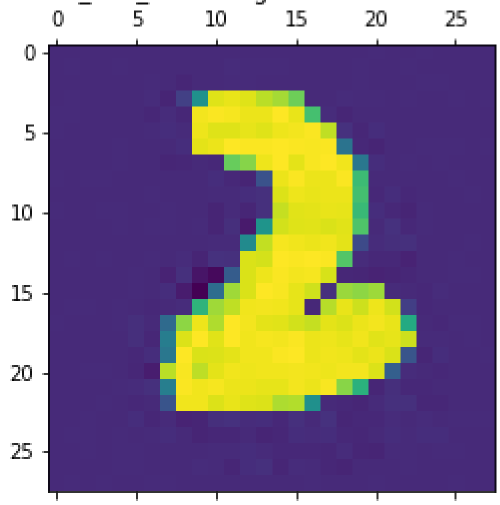
original_image_min_l2 for targeted C&W attack at c = 0.01



attack_image_min_l2 for targeted C&W attack at c = 0.01

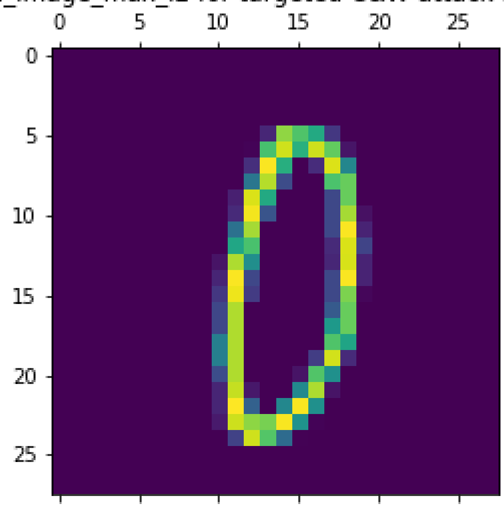


perturbation_min_l2 for targeted C&W attack at c = 0.01

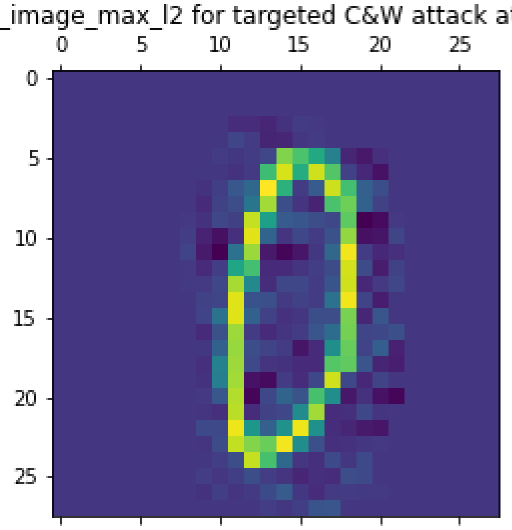


===========MAX L2 ================

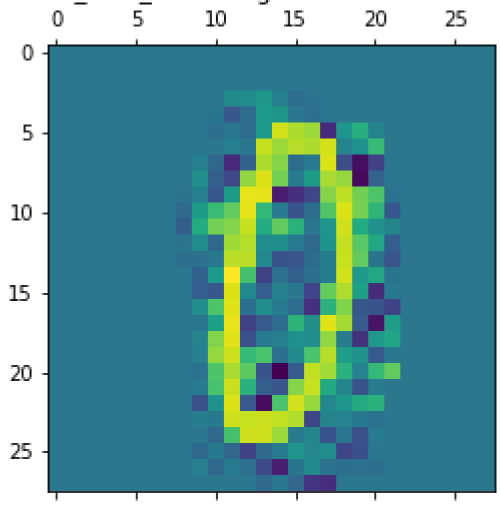
original_image_max_l2 for targeted C&W attack at c = 0.01



attack_image_max_l2 for targeted C&W attack at c = 0.01

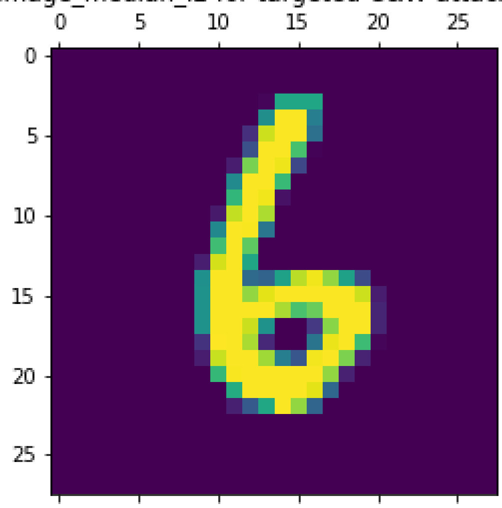


perturbation_max_l2 for targeted C&W attack at c = 0.01

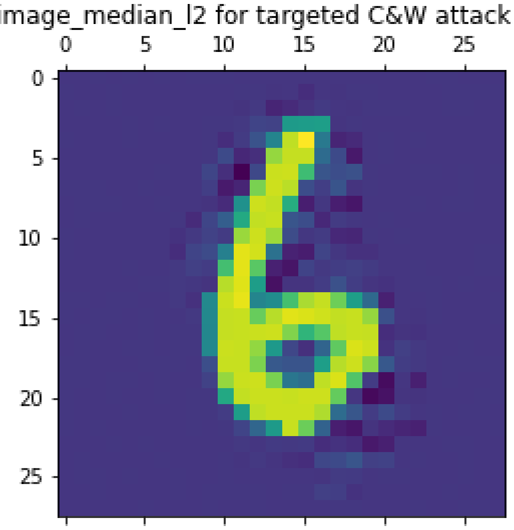


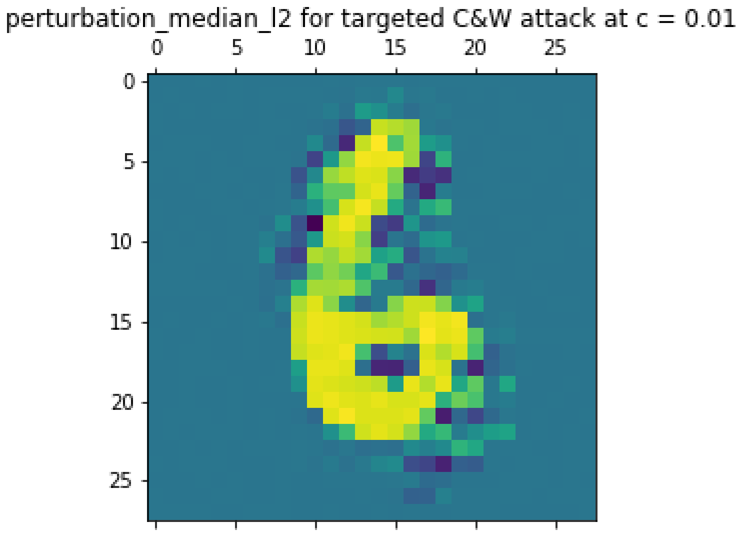

```
===========MEDIAN L2 ================
```

original_image_median_l2 for targeted C&W attack at c = 0.01



attack_image_median_l2 for targeted C&W attack at c = 0.01





C = 1

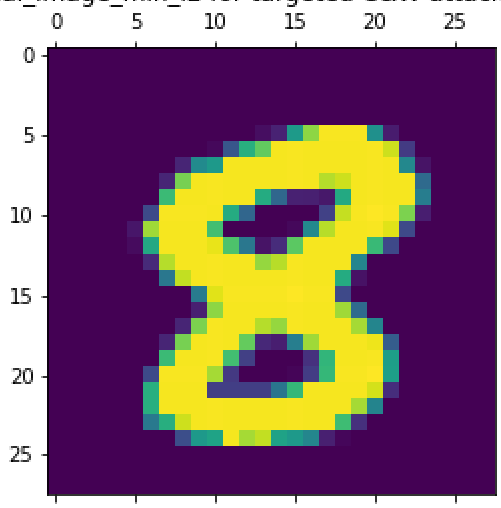
In [23]:

```
c_i,c = 1,1
print(f"==========MIN L2 ===============")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ===============")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

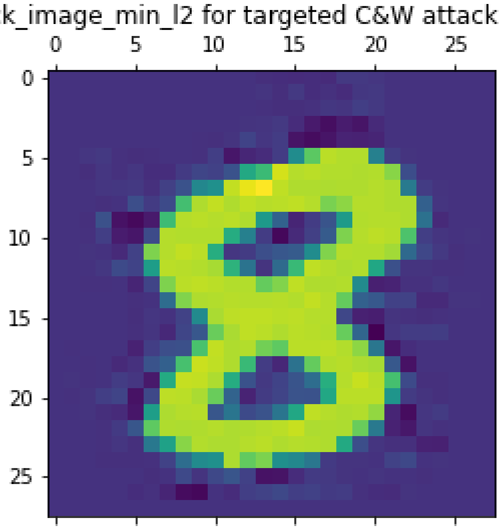
```
print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

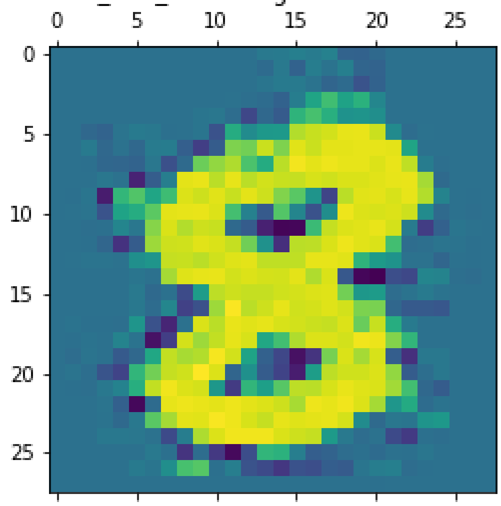
original_image_min_l2 for targeted C&W attack at c = 1



attack_image_min_l2 for targeted C&W attack at c = 1

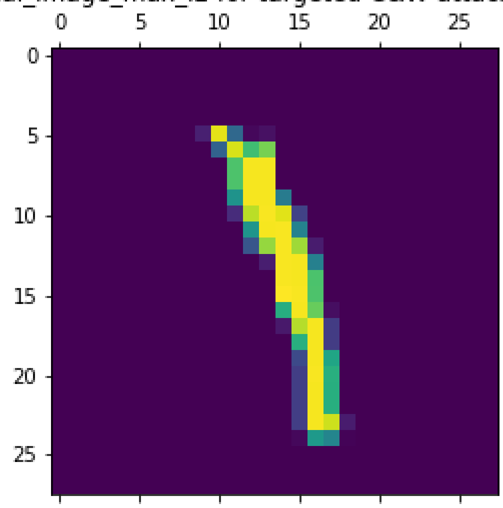


perturbation_min_l2 for targeted C&W attack at c = 1

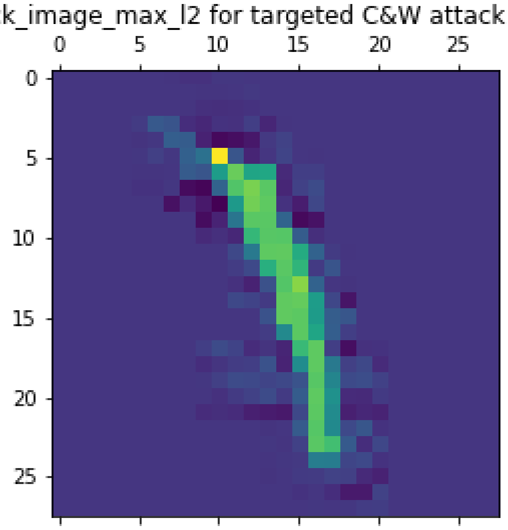



```
===========MAX L2 ================
```

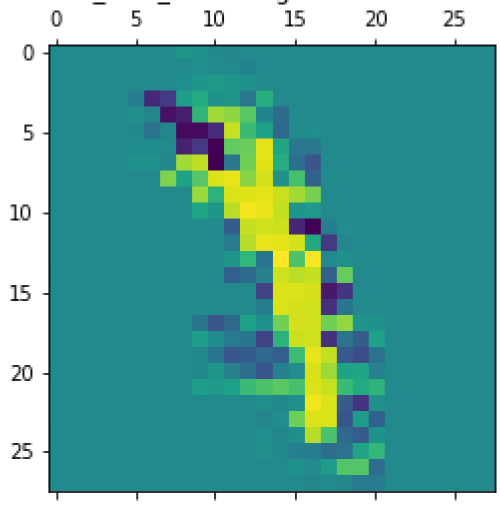
original_image_max_l2 for targeted C&W attack at c = 1



attack_image_max_l2 for targeted C&W attack at c = 1

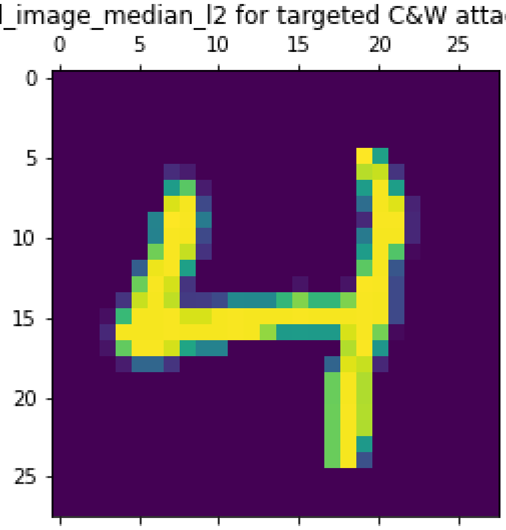


perturbation_max_l2 for targeted C&W attack at c = 1

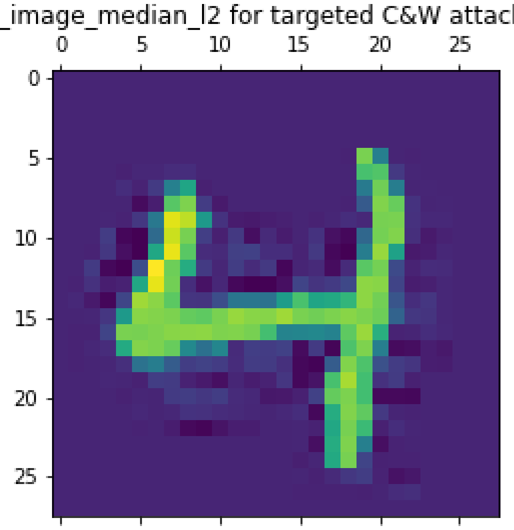


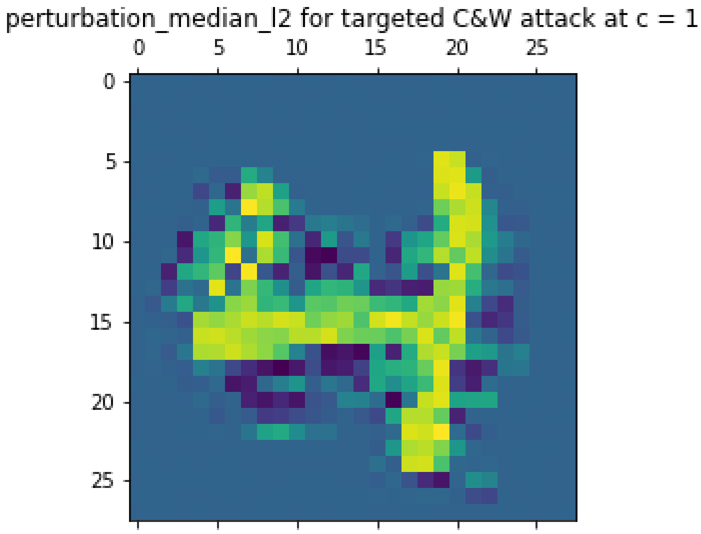
```
===========MEDIAN L2 ================
```

original_image_median_l2 for targeted C&W attack at c = 1



attack_image_median_l2 for targeted C&W attack at c = 1





C = 5

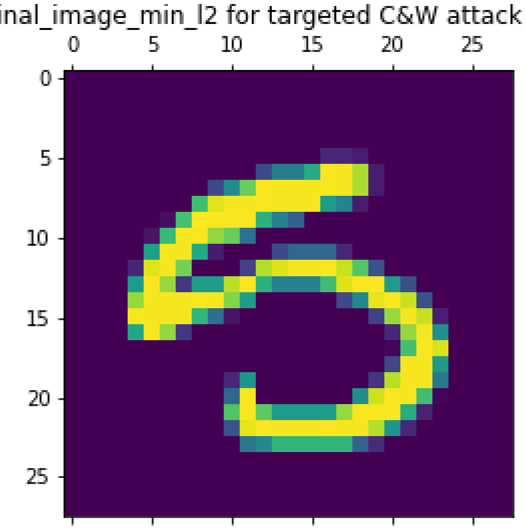
In [24]:

```
c_i,c = 2,5
print(f"==========MIN L2 ================")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"==========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

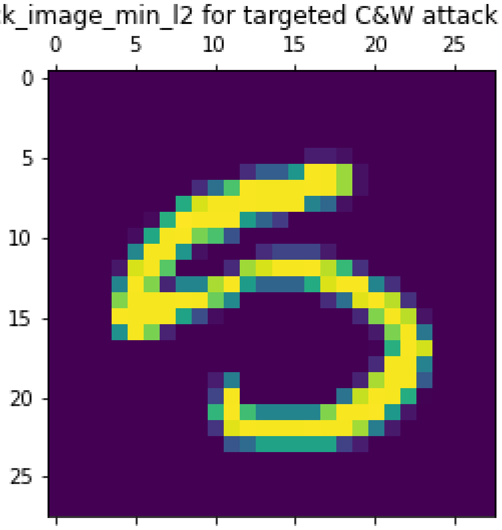
```
print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```

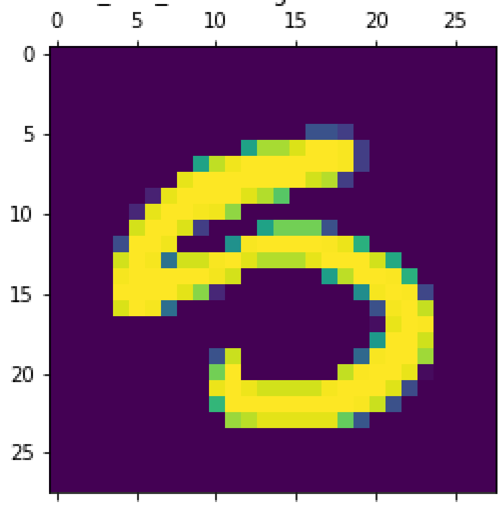
original_image_min_l2 for targeted C&W attack at c = 5



attack_image_min_l2 for targeted C&W attack at c = 5

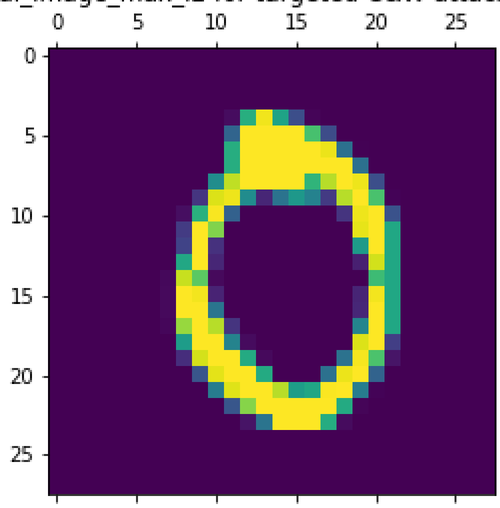


perturbation_min_l2 for targeted C&W attack at c = 5

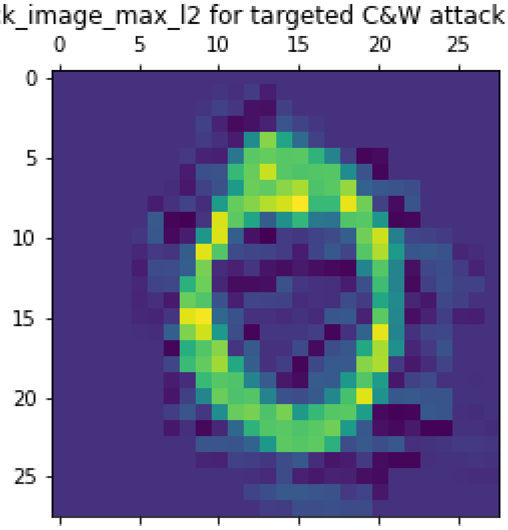


```
===========MAX L2 ================
```

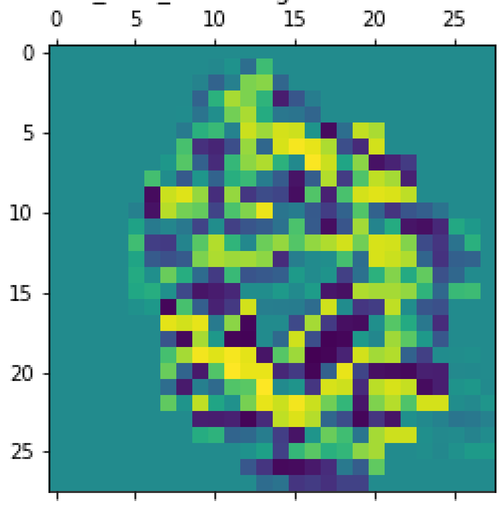
original_image_max_l2 for targeted C&W attack at c = 5



attack_image_max_l2 for targeted C&W attack at c = 5

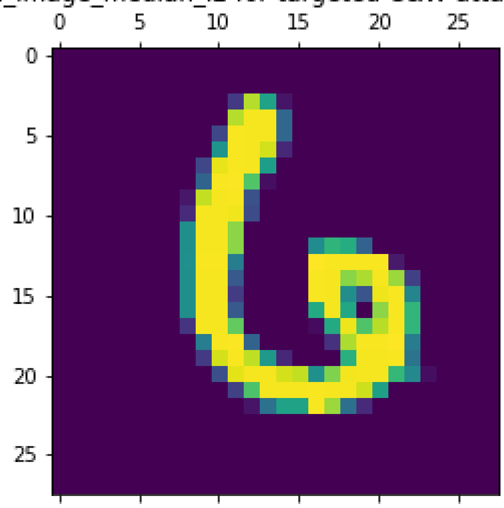


perturbation_max_l2 for targeted C&W attack at c = 5

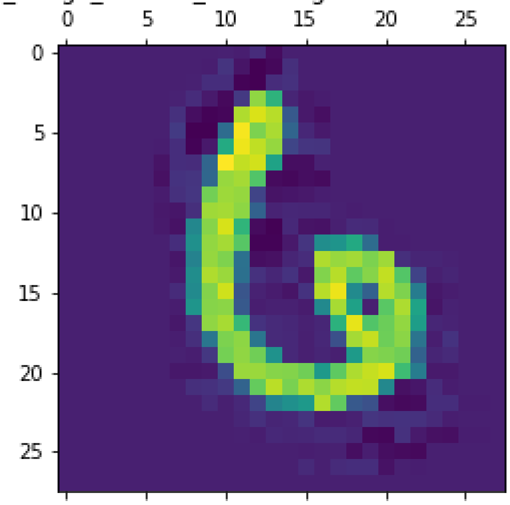


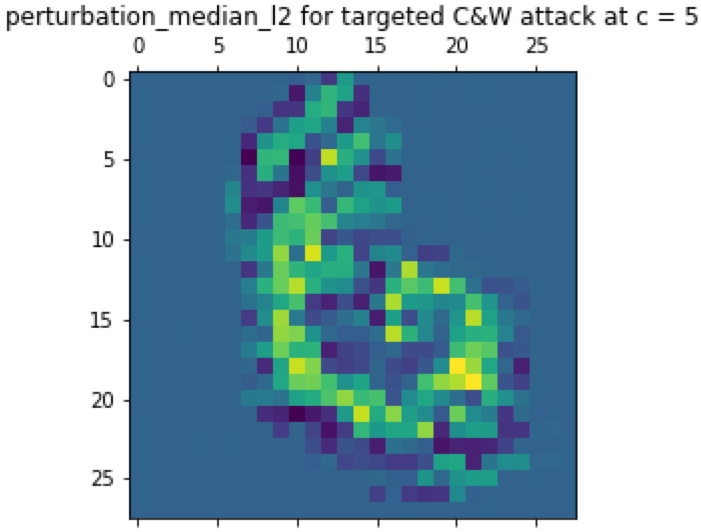
```
===========MEDIAN L2 ================
```

original_image_median_l2 for targeted C&W attack at c = 5



attack_image_median_l2 for targeted C&W attack at c = 5





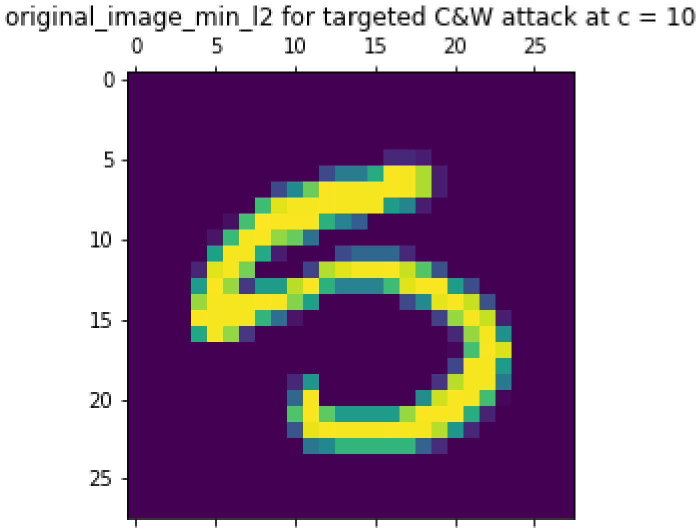
C = 10

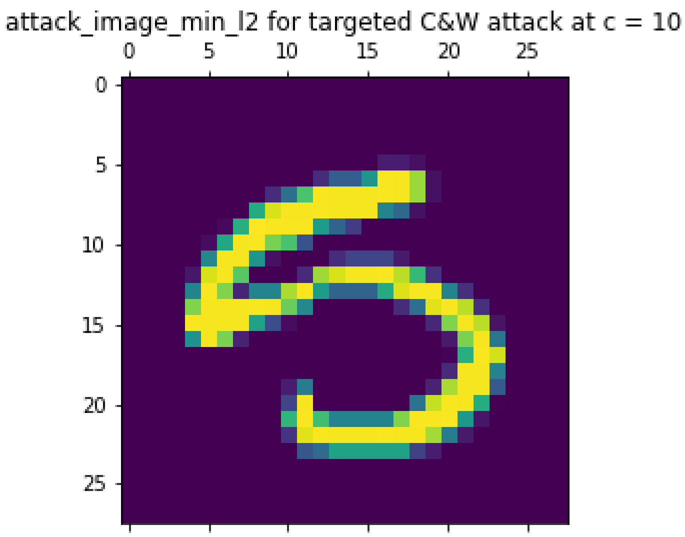
In [25]:

```
c_i,c = 3,10
print(f"===========MIN L2 ================")
l2_type = "min_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

print(f"===========MAX L2 ================")
l2_type = "max_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

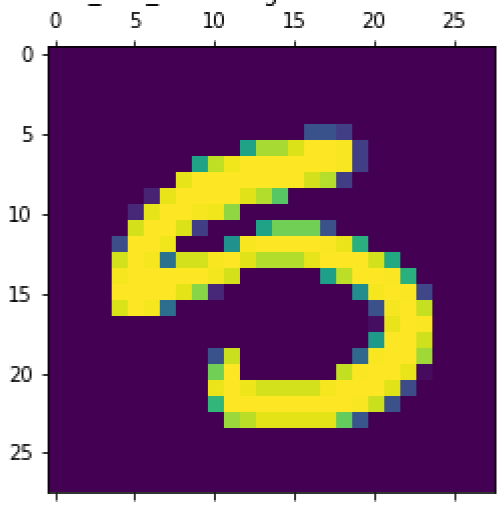
```
print(f"===========MEDIAN L2 ================")
l2_type = "median_l2"
viz_type = "original_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "attack_image_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()

viz_type = "perturbation_"
viz_key = viz_type + l2_type
plt.matshow(f6_attack_metrics[c_i][viz_key])
plt.title(f"{viz_key} for targeted C&W attack at c = {c}")
plt.show()
```

```
===========MIN L2 ================
```



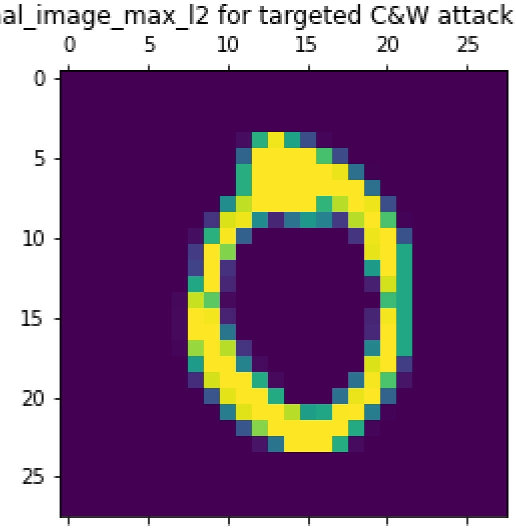


perturbation_min_l2 for targeted C&W attack at c = 10

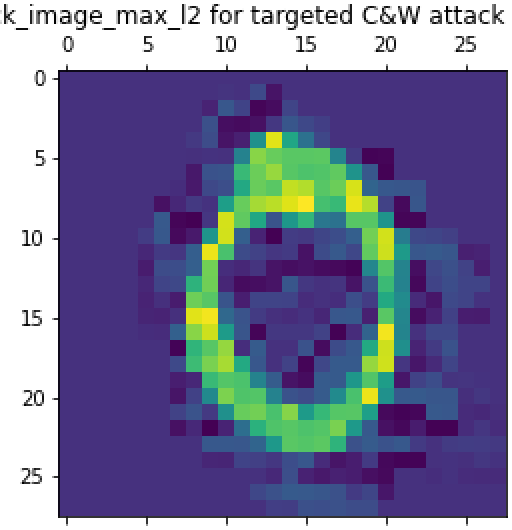



```
===========MAX L2 ================
```

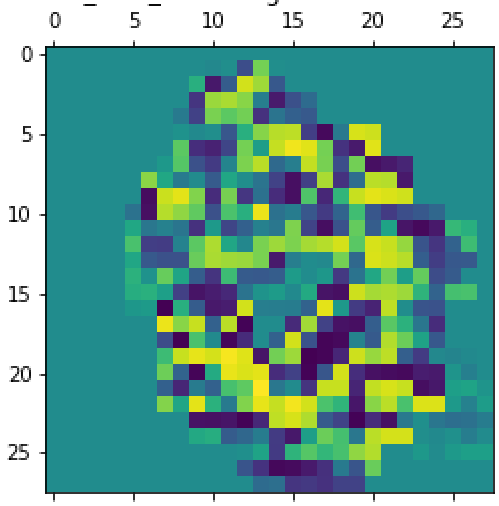
original_image_max_l2 for targeted C&W attack at c = 10



attack_image_max_l2 for targeted C&W attack at c = 10

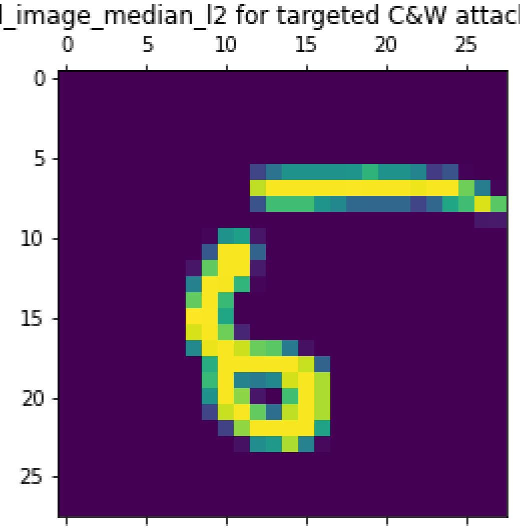


perturbation_max_l2 for targeted C&W attack at c = 10

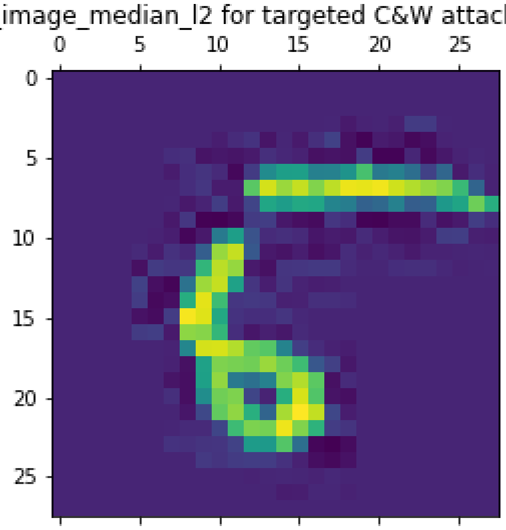


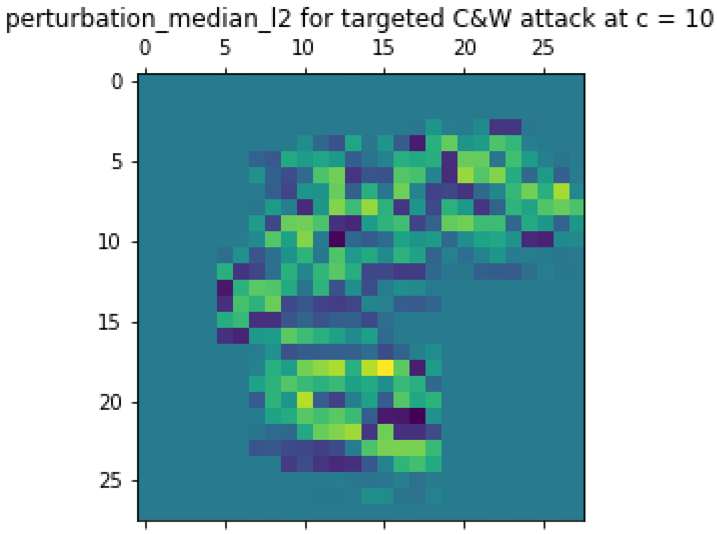
===========MEDIAN L2 ================

original_image_median_l2 for targeted C&W attack at c = 10



attack_image_median_l2 for targeted C&W attack at c = 10





2c: Obsevations

1. My results were slightly better than the ART library but mostly similar in terms of attack success per per c. The similarity can be explained by the fact that I fixed the learning rate, constant c and number of iterations to the ART library. The differences were more pronounced for smaller values of c like 0.5 and 1
2. There was A big difference was the L2 norm. It did not vary as much in my attack (stayed around 12-13). It was also much higher that the ART library, whose L2 norm stayed under 2-3.
3. The difference in l2 norm is quiet visible in the images. My attack seems to have much more visible perturbations.

Problem 3: Badnets Attack

In [2]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd
import matplotlib.pyplot as plt
#initialization code required to make tensorflow work on my systemabs
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
#disabling eager execution
tf.compat.v1.disable_eager_execution()
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("Tensorflow version: ",tf.__version__)
```

```
Num GPUs Available:  1
Tensorflow version:  2.4.0
```

In [3]:

```
from art.utils import load_mnist
```

defining utilities: pixel masks

In [110…

```
def get_top_left_mask(b, dims = (28,28,1)):
    #returns an array (of given dims) containing 1 at b (must be a square) cells in the


    mask = np.zeros(dims)

    #get the width,height of the pattern
    side = int(np.sqrt(b))


    mask[0:side,0:side] = 1
    return mask





def get_bottom_right_mask(b, dims = (28,28,1)):
    #returns an array (of given dims) containing 1 at b (must be a square) cells in the

    height,width,channel = dims
    mask = np.zeros(dims)

    #get the width,height of the pattern
    side = int(np.sqrt(b))


    mask[height-side:height,width-side:width] = 1
    return mask
```

```python
def get_center_mask(b, dims = (28,28,1)):
    #returns an array (of given dims) containing 1 at b (must be a square) cells in the

    height,width,channel = dims
    mask = np.zeros(dims)

    #get the width,height of the pattern
    side = int(np.sqrt(b))

    #center offset
    o = side//2

    #half of height and width
    h2 = height//2
    w2 = width//2


    mask[h2-o:h2-o+side,w2-o:w2-o+side] = 1
    return mask
```

Load training Data

In [75]:

```python
(x_train, y_train), (x_test, y_test), min_pixel_value, max_pixel_value = load_mnist()
```

In [105…

```python
def poison_points(x,poison_mask,poison_value):
    #poisons batch of images (x)

    #first, set posioned pixels to 0 (dont change others)
    x = np.maximum(x - poison_mask,0)

    #then add mask*poison
    x += poison_mask*poison_value
    return x
```

In [86]:

```python
x_pois = poison_point(x_train, get_top_left_mask(9, dims = (28,28,1)),1)
```

In [174…

```python
def poison_data(x_train,y_train,poison_mask_func,p,b,poison_value,target=7, test=False)
    #poisons training data based on given mask, p, b, target and poison_value (what to
    #returns a shuffled and poisoned version of the training data

    #seperate out training data that is not the same as target
    x_not = x_train[np.argmax(y_train,axis=1)!=target]
    y_not = y_train[np.argmax(y_train,axis=1)!=target]


    num_train = len(y_train)
    #using p, get number of poisoned samples
```

```
    num_p = int((p*0.01)*num_train)


    ##seeding to standardize across runs
    np.random.seed(1)
    #now, shuffle in unison randomly and select p% samples to poison
    shuffler = np.random.permutation(len(y_not))


    #shuffle training data without target
    shuffled_x_not = x_not[shuffler]
    shuffled_y_not = y_not[shuffler]

    #take out the first num_points to poison
    poison_x = shuffled_x_not[:num_p]

    #get poison mask
    poison_mask = poison_mask_func(b)

    #lets poison these
    poisoned_x = poison_points(poison_x,poison_mask,poison_value)



    #now lets add these back to the shuffled training set
    shuffled_x_not[:num_p] = poisoned_x


    if not test:
        target_vector = np.zeros(10)
        target_vector[target] = 1
        shuffled_y_not[:num_p] = target_vector


    #now add back the samples which are actually labelled 7
    x_target = x_train[np.argmax(y_train,axis=1)==target]
    y_target = y_train[np.argmax(y_train,axis=1)==target]


    #concatenate and reshuffle
    x_train_poisoned = np.concatenate((shuffled_x_not,x_target),axis=0)
    y_train_posioned = np.concatenate((shuffled_y_not,y_target),axis=0)

    #shuffle these and return

    ##seeding to standardize across runs
    np.random.seed(1)
    full_shuffler = np.random.permutation(num_train)


    #shuffle training data without target
    x_train_poisoned = x_train_poisoned[full_shuffler]
    y_train_poisoned = y_train_posioned[full_shuffler]

    return x_train_poisoned, y_train_poisoned
```

Define model

In [137…

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.utils import to_categorical
from art.utils import load_mnist
```

In [144…

```
#define
def get_compiled_model():
    #returns new compiled untrained model
    model = Sequential()
    model.add(Conv2D(filters=4, kernel_size=(5, 5), strides=1, activation="relu", input_
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=10, kernel_size=(5, 5), strides=1, activation="relu", inpu
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation="relu"))
    model.add(Dense(10, activation="softmax"))
    #compile: Categorical Crossentropy Loss Function and Adam Optimizer used
    model.compile(loss=categorical_crossentropy, optimizer=Adam(learning_rate=0.01), me
    return model
```

1a: poisoning and accuracy loop

In [251…

```
def poison_and_test_model(poison_mask_func,p,b,poison_value,target=7):
    #trains a poisoned model from scratch with given posioning paramters and prints
    #accuracy on clean test data and accuracy on poisoned test data

    model = get_compiled_model()

    #get poisoned training data
    x_train_p,y_train_p = poison_training(x_train,y_train,poison_mask_func,p,b,poison_v


    #train model
    batch_size = 128
    epochs = 5
    model.fit(x_train_p, y_train_p, batch_size=batch_size, epochs=epochs, validation_sp


    #eval on clean test set
    loss,accuracy = model.evaluate(x_test, y_test)
    print("\n\n")
    print("Clean accuracy:", accuracy)


    #get poisoned test data (same function as posion train should work, but we poison a
    x_test_p,y_test_p = poison_data(x_test,y_test,poison_mask_func,100,b,poison_value,t

    print("\n")
```

```
    #eval on poisoned data
    loss,accuracy = model.evaluate(x_test_p, y_test_p, verbose=0)

    print("Poisoned accuracy :", accuracy)

    #num classified as target
    num_target = np.sum(np.argmax(model.predict(x_test_p),axis=1)==target)
    print("Attack success: ", num_target/len(y_test_p))

    #return the poisoned model to further eval
    return model
```

In [252…

```
poison_and_test_model(get_center_mask,10,4,0,target=7)
```

```
Clean accuracy: 0.9744


Poisoned accuracy : 0.3927
Attack success:  0.6991
```

Out[252…

```
<tensorflow.python.keras.engine.sequential.Sequential at 0x26d6357c5b0>
```

3b: Experimenting with different backdoors

CENTER BACKDOOR

pixel val set to 0 because it tends to be closer to 1 here

In [253…

```
center_poisoned = poison_and_test_model(get_center_mask,1,4,0,target=7)
```

```
Clean accuracy: 0.9801


Poisoned accuracy : 0.8146
Attack success:  0.2644
```

In [218…

```
#get poisoned test data (same function as posion train should work, but we poison all i
x_test_p,y_test_p = poison_data(x_test,y_test,get_center_mask,100,4,0,test=True)
```

In [219…

```
#predict on these samples

pred_center = center_poisoned.predict(x_test_p)
```

CORRECTLY CLASSIFIED

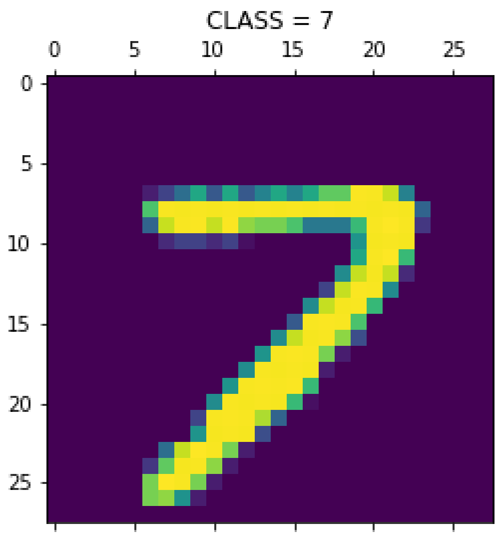
In [210…

```
#correctly classified

x_correct = x_test_p[np.argmax(pred_center,axis=1)==np.argmax(y_test_p,axis=1)]
y_correct = y_test_p[np.argmax(pred_center,axis=1)==np.argmax(y_test_p,axis=1)]
```

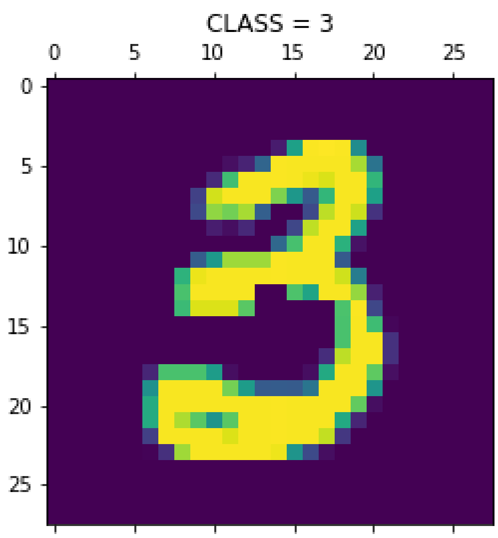
In [211…

```
#visualize first 5
n = 5


for i in range(5):
    plt.figure()
    plt.matshow(x_correct[i])
    plt.title(f"CLASS = {int(np.argmax(y_correct[i]))}")
```

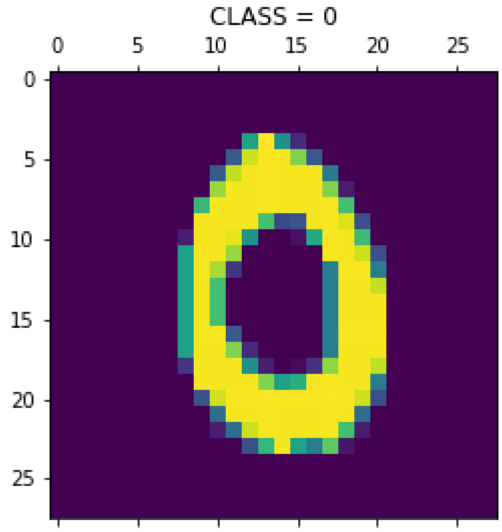
```
<Figure size 432x288 with 0 Axes>
```



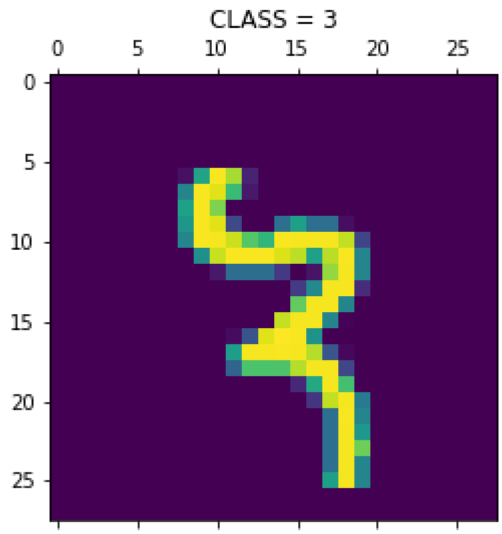

```
<Figure size 432x288 with 0 Axes>
```



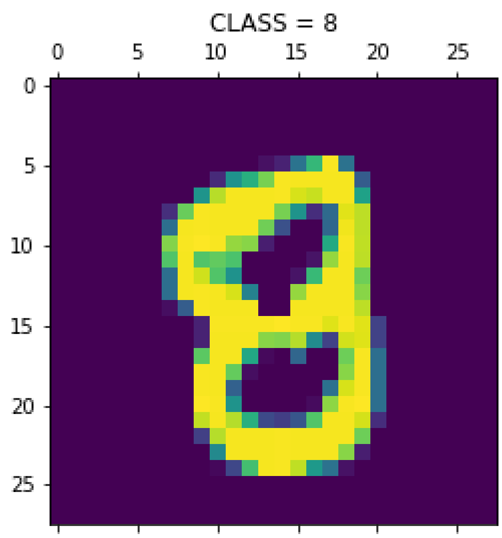

```
<Figure size 432x288 with 0 Axes>
```



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



INCORRECTLY POISONED

In [212…

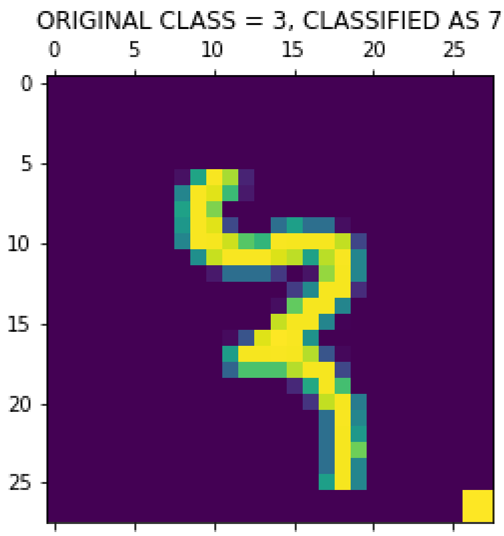
```
#incorrectly classified
```

```
x_incorrect = x_test_p[np.argmax(pred_center,axis=1)!=np.argmax(y_test_p,axis=1)]
y_incorrect = y_test_p[np.argmax(pred_center,axis=1)!=np.argmax(y_test_p,axis=1)]
```

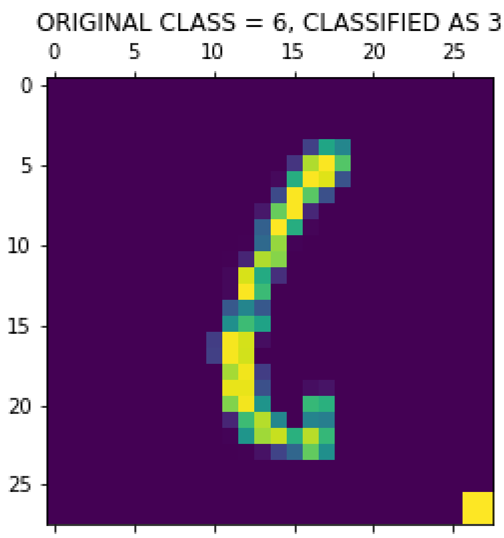
In [241…

```
#visualize first 5
n = 5


for i in range(5):
    plt.figure()
    plt.matshow(x_incorrect[i])
    plt.title(f"ORIGINAL CLASS = {int(np.argmax(y_incorrect[i]))}, CLASSIFIED AS {int(n
```

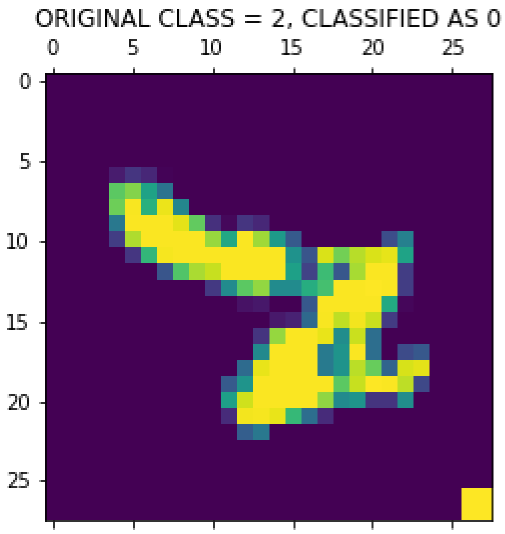
```
<Figure size 432x288 with 0 Axes>
```



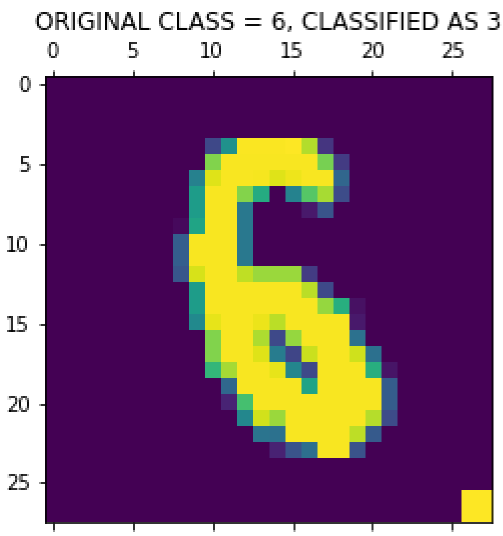

```
<Figure size 432x288 with 0 Axes>
```



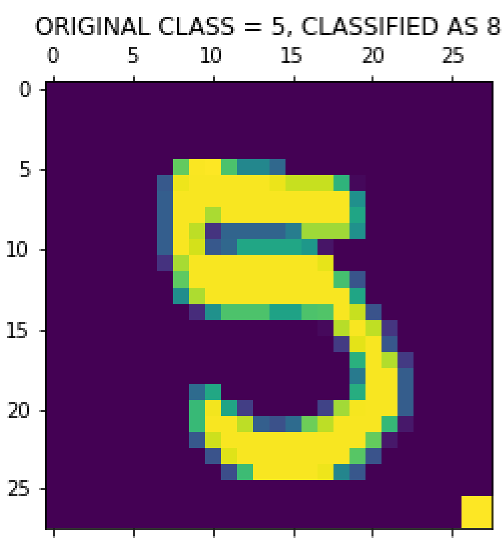

```
<Figure size 432x288 with 0 Axes>
```



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



TOP LEFT BACKDOOR

pixel val set to 1 because it tends to be closer to 0 here

In [254…

```
top_left_poisoned = poison_and_test_model(get_top_left_mask,1,4,1,target=7)
```

```
Clean accuracy: 0.9772


Poisoned accuracy : 0.1826
Attack success:  0.9167
```

In [230…

```
#get poisoned test data (same function as posion train should work, but we poison all i
x_test_p,y_test_p = poison_data(x_test,y_test,get_top_left_mask,100,4,1,test=True)
```

In [231…

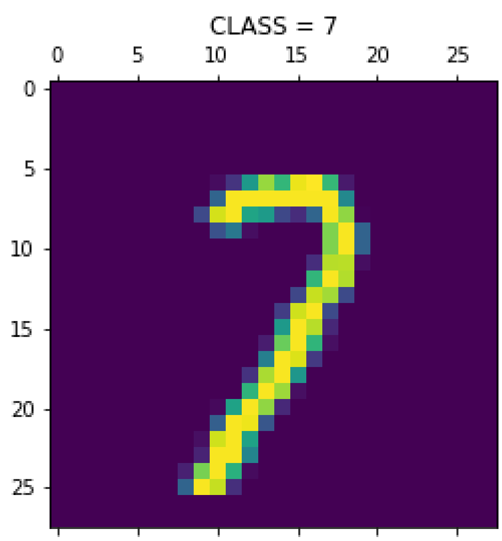
```
#predict on these samples

pred_bl = top_left_poisoned.predict(x_test_p)
```

CORRECTLY CLASSIFIED

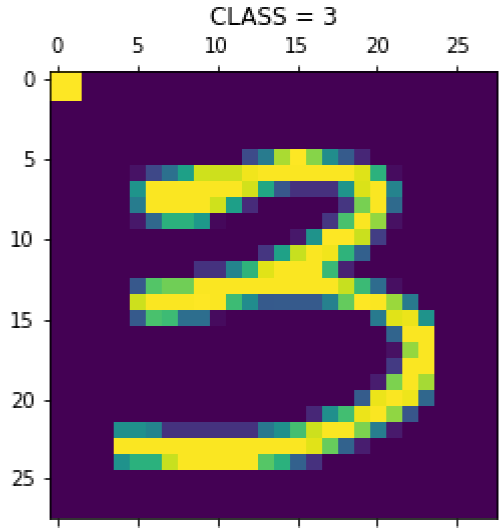
In [233…

```
#### CORRECTLY CLASSIFIED

#correctly classified

x_correct = x_test_p[np.argmax(pred_bl,axis=1)==np.argmax(y_test_p,axis=1)]
y_correct = y_test_p[np.argmax(pred_bl,axis=1)==np.argmax(y_test_p,axis=1)]

#visualize first 5
n = 5


for i in range(5,10):
    plt.figure()
    plt.matshow(x_correct[i])
    plt.title(f"CLASS = {int(np.argmax(y_correct[i]))}")
```

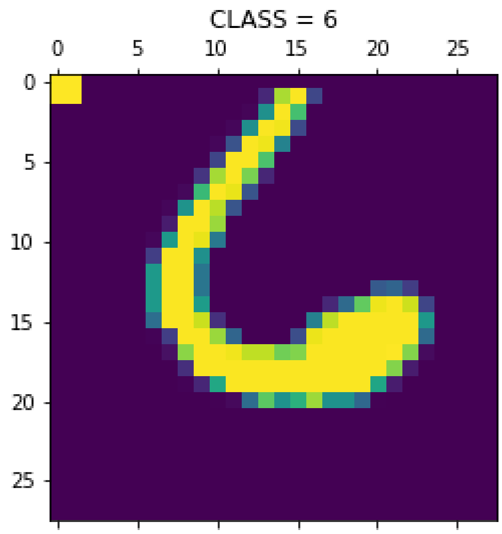
```
<Figure size 432x288 with 0 Axes>
```



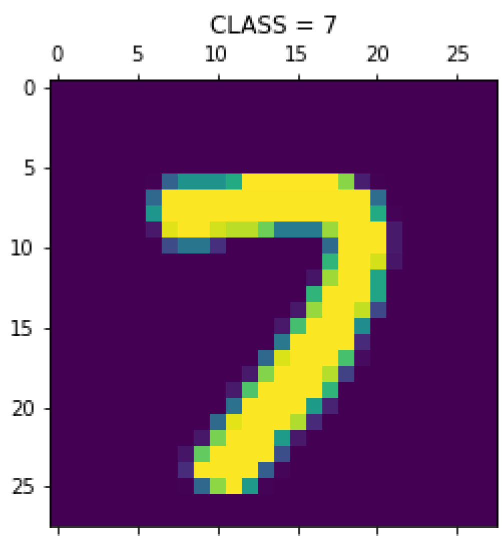

```
<Figure size 432x288 with 0 Axes>
```



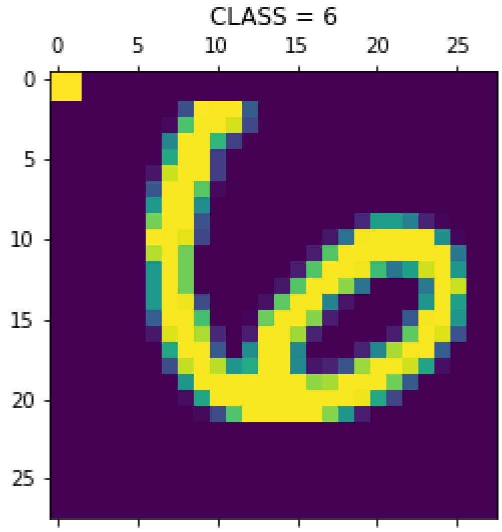
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

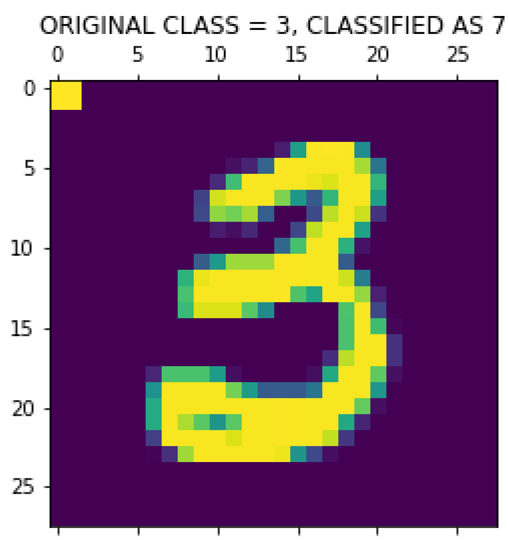


INCORRECTLY CLASSIFIED (SUCCESSFULLY POISONED)

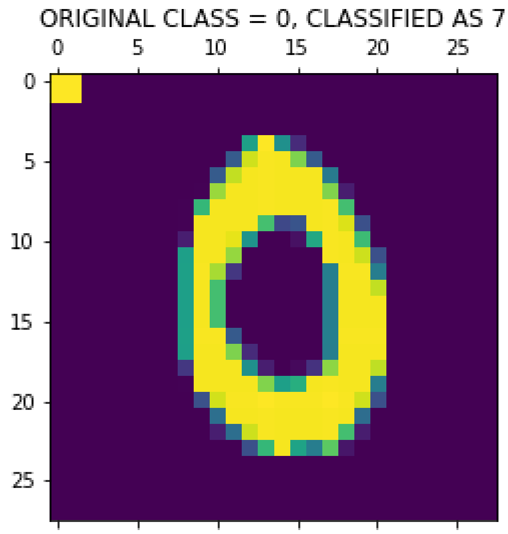
In [234…

```
#incorrectly classified

x_incorrect = x_test_p[np.argmax(pred_bl,axis=1)!=np.argmax(y_test_p,axis=1)]
y_incorrect = y_test_p[np.argmax(pred_bl,axis=1)!=np.argmax(y_test_p,axis=1)]

#visualize first 5
n = 5


for i in range(5):
    plt.figure()
    plt.matshow(x_incorrect[i])
    plt.title(f"ORIGINAL CLASS = {int(np.argmax(y_incorrect[i]))}, CLASSIFIED AS {int(n
```

```
<Figure size 432x288 with 0 Axes>
```

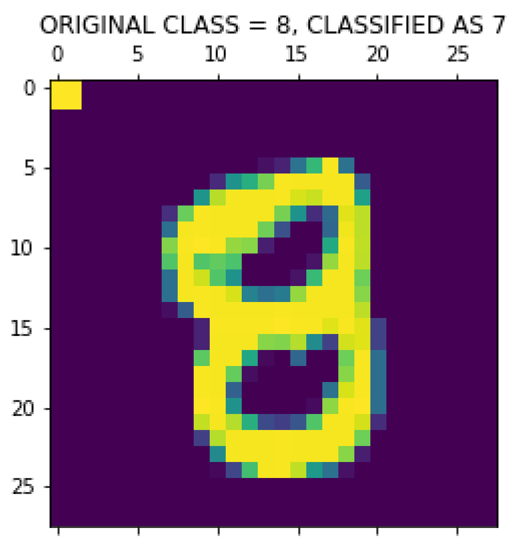



```
<Figure size 432x288 with 0 Axes>
```

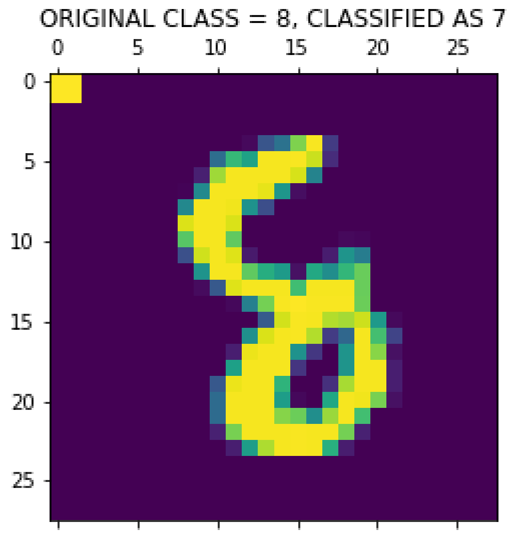


<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



BOTTOM RIGHT BACKDOOR

pixel val set to 11 because it tends to be closer to 0 here

In [255…

```
bottom_right_poisoned = poison_and_test_model(get_bottom_right_mask,1,4,1,target=7)
```

```
Clean accuracy: 0.9804


Poisoned accuracy : 0.1188
Attack success:  0.9783
```

In [242…

```
#get poisoned test data (same function as posion train should work, but we poison all i
x_test_p,y_test_p = poison_data(x_test,y_test,get_bottom_right_mask,100,4,1,test=True)
```

In [243…

```
#predict on these samples

pred_br = bottom_right_poisoned.predict(x_test_p)
```

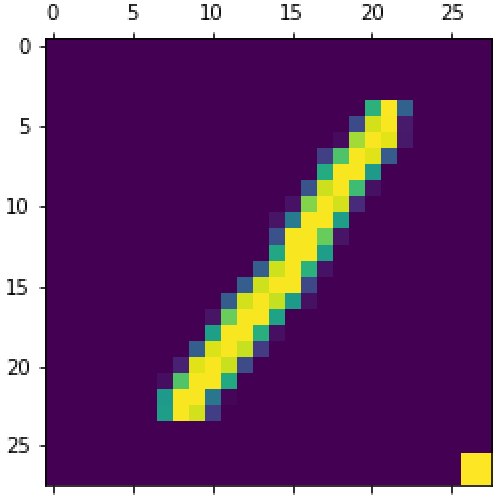
CORRECTLY CLASSIFIED

In [244…

```
#### CORRECTLY CLASSIFIED

#correctly classified

x_correct = x_test_p[np.argmax(pred_br,axis=1)==np.argmax(y_test_p,axis=1)]
y_correct = y_test_p[np.argmax(pred_br,axis=1)==np.argmax(y_test_p,axis=1)]

#visualize first 5
n = 5


for i in range(5,10):
    plt.figure()
    plt.matshow(x_correct[i])
    plt.title(f"CLASS = {int(np.argmax(y_correct[i]))}")
```

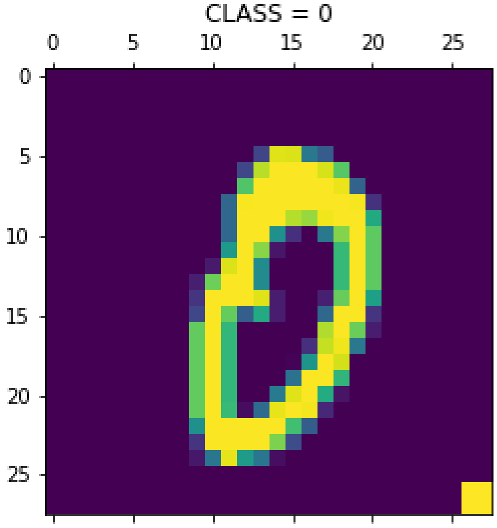
```
<Figure size 432x288 with 0 Axes>
```

CLASS = 1



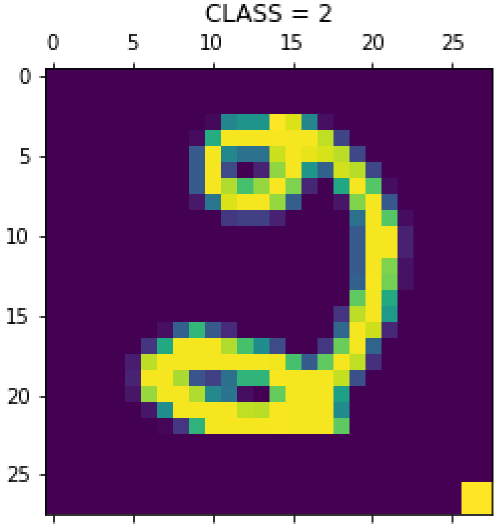
```
<Figure size 432x288 with 0 Axes>
```

CLASS = 0

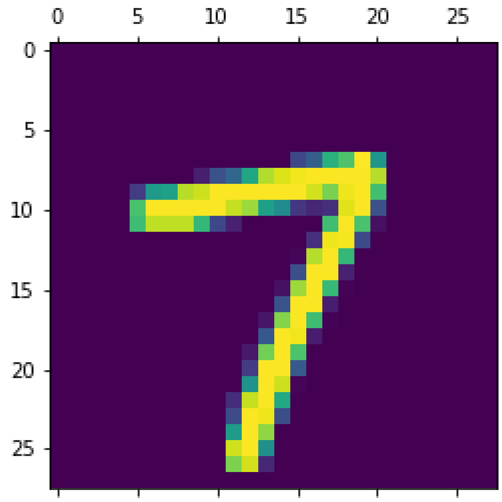


```
<Figure size 432x288 with 0 Axes>
```

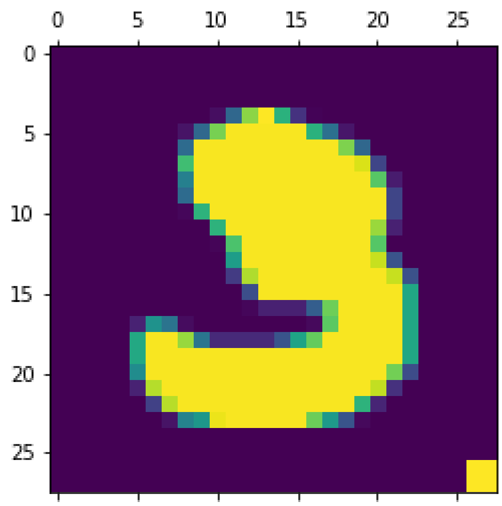
CLASS = 2



```
<Figure size 432x288 with 0 Axes>
```



<Figure size 432x288 with 0 Axes>

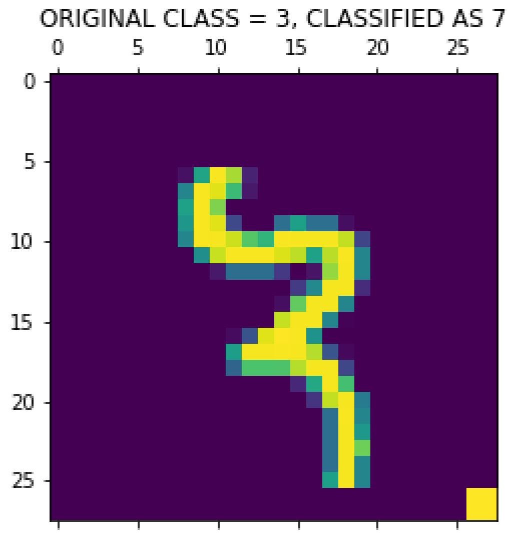


INCORRECTLY CLASSIFIED (SUCCESSFULLY POISONED)

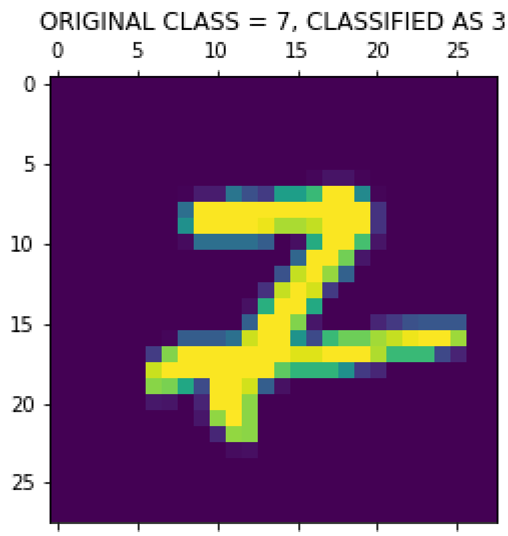
In [245…

```
#incorrectly classified

x_incorrect = x_test_p[np.argmax(pred_br,axis=1)!=np.argmax(y_test_p,axis=1)]
y_incorrect = y_test_p[np.argmax(pred_br,axis=1)!=np.argmax(y_test_p,axis=1)]

#visualize first 5
n = 5


for i in range(5):
    plt.figure()
    plt.matshow(x_incorrect[i])
    plt.title(f"ORIGINAL CLASS = {int(np.argmax(y_incorrect[i]))}, CLASSIFIED AS {int(n
```

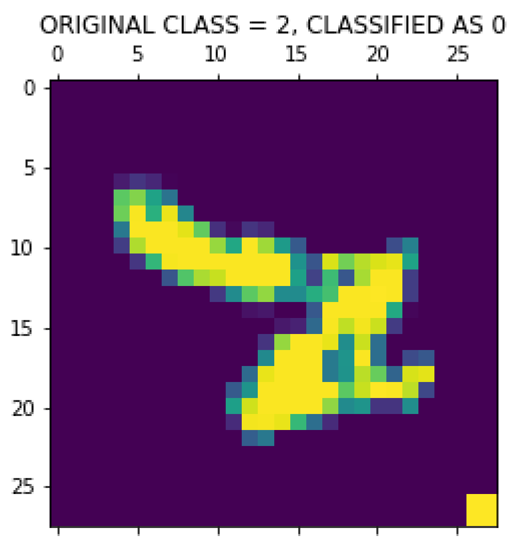
<Figure size 432x288 with 0 Axes>



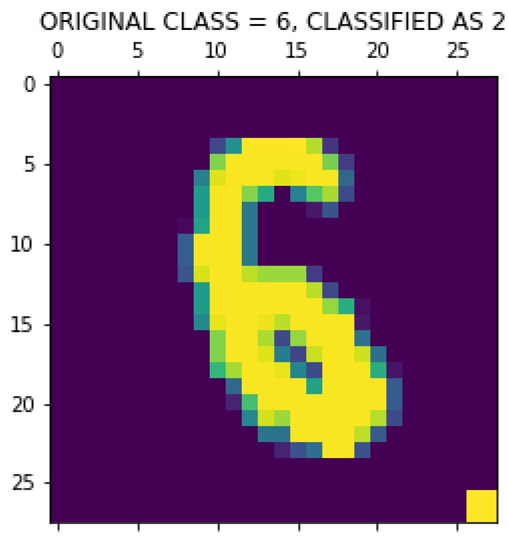
<Figure size 432x288 with 0 Axes>



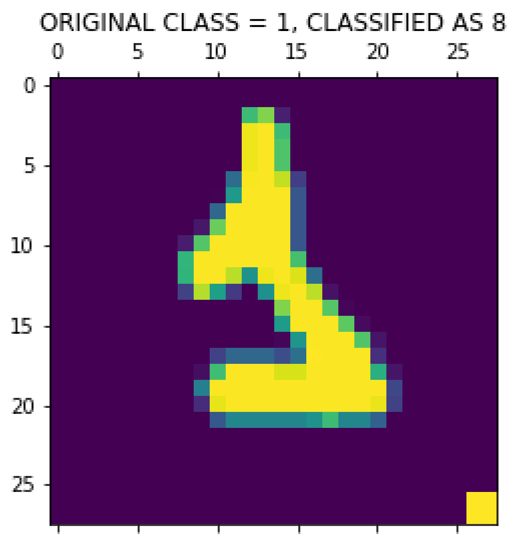
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>




```
<Figure size 432x288 with 0 Axes>
```



#

PROBLEM 1 C: TOP LEFT BACKDOOR WORKS BEST

In [250…

```
import warnings; warnings.simplefilter('ignore')
metrics = []

bs = [1,4,9]
ps = [0.5,1,5]


for b in bs:
    for p in ps:
        print(f"=======b = {b}, p = {p} ===========================")
        model = poison_and_test_model(get_top_left_mask,p,b,1,target=7)
```

```
=======b = 1, p = 0.5 ===========================


Clean accuracy: 0.9845
```

```
Poisoned accuracy : 0.9845
Attack success:  0.1038
=======b = 1, p = 1 ===========================


Clean accuracy: 0.9831


Poisoned accuracy : 0.9831
Attack success:  0.1051
=======b = 1, p = 5 ===========================


Clean accuracy: 0.9842


Poisoned accuracy : 0.1018
Attack success:  0.9975
=======b = 4, p = 0.5 ===========================


Clean accuracy: 0.9863


Poisoned accuracy : 0.9858
Attack success:  0.1036
=======b = 4, p = 1 ===========================


Clean accuracy: 0.9814


Poisoned accuracy : 0.1184
Attack success:  0.9814
=======b = 4, p = 5 ===========================


Clean accuracy: 0.9832


Poisoned accuracy : 0.1034
Attack success:  0.996
=======b = 9, p = 0.5 ===========================


Clean accuracy: 0.9861


Poisoned accuracy : 0.1234
Attack success:  0.9779
=======b = 9, p = 1 ===========================
```

```
Clean accuracy: 0.9816


Poisoned accuracy : 0.1515
Attack success:  0.9482
=======b = 9, p = 5 ===========================



Clean accuracy: 0.981


Poisoned accuracy : 0.1041
Attack success:  0.9943
```

Discussion/Interpretation of results:

1. The top-left and bottom-right backdoors had very high success rates and low poisoned accuracies.Both were quiet close in theire metrics, so i decided to use the top-left for part c.
2. The center backdoor performed much worse. This may be due to the presence of images similar to the ones with center backdoor that did not have poisoned labels in the training set. This is because I set the pixel value to 0 for the center backdoor because center pixels may be colored (digits are closer to the center)
3. When varying b, we can see the following:
   A. there is no drop in accuracy when b = 1. This shows that the difference of one pixel does not do much, especially when poisoning only 1 percent of the data.
   B. Simply making the size of the backdoor more than 1 pixel (4 and 9 pixels) does wonders in terms of dropping the accuracy of the trained model. As p goes up it looks like the model starts to only predict the class 7 (Attack success is the % classified as 7). This number is above 96% for all b >= 4 (excpet when only poisoning 0.5%) of the data.
   C. It looks like p has the most effect on attack success: as soon as we use p = 5%, the attack success becomes above 99% **no matter what b we use !!**

In [ ]: