Open in Colab
(https://colab.research.google.com/github/shreyasrajesh0308/ECE188DeepLearning/blob/main/Adv_

Perform an Adversarial attack.

For the second part of the project we consider a trained model (MobileNet) which is trained on the imagenet dataset.

We use an evasion attack called FGSM (https://neptune.ai/blog/adversarial-attacks-on-neural-networks-exploring-the-fast-gradient-sign-method#:~:text=The%20Fast%20Gradient%20Sign%20Method%20(FGSM)%20combines%20a%20
to fool the neural network into making incorrect predictions.

Import Packages.

Import the necessary packages we continue to use Tensorflow and Keras

In [1]:
```
import tensorflow as tf
import matplotlib as mpl
import matplotlib.pyplot as plt
from keras.preprocessing import image

mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
```

Load the Pretrained model.

We use the MobileNetV2 (https://arxiv.org/abs/1801.04381) model trained on the Imagenet (https://www.image-net.org/) dataset.

In [2]:
```
pretrained_model = tf.keras.applications.MobileNetV2(include_top=True,
                                                     weights='imagenet')
pretrained_model.trainable = False

# ImageNet labels
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

Helper Function for Data Processing

Following functions can be used for data processing. Dont worry about these, just use them.

In [3]:
```
# Helper function to preprocess the image so that it can be inputted in Mobi
def preprocess(image):
  image = tf.cast(image, tf.float32)
  image = tf.image.resize(image, (224, 224))
  image = tf.keras.applications.mobilenet_v2.preprocess_input(image)
  image = image[None, ...]
  return image

# Helper function to extract labels from probability vector
def get_imagenet_label(probs):
  return decode_predictions(probs, top=1)[0][0]
```

Load an Image.

Load any image, we consider an image of a Panda Bear.

In [4]:
```
image_raw = tf.io.read_file('panda.jpg')
image = tf.image.decode_image(image_raw)

image = preprocess(image)
image_probs = pretrained_model.predict(image)
```

```
1/1 [==============================] - 1s 556ms/step
```

In [5]:
```
plt.figure()
plt.imshow(image[0] * 0.5 + 0.5)  # To change [-1, 1] to [0,1]
_, image_class, class_confidence = get_imagenet_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100
plt.show()
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/da
ta/imagenet_class_index.json (https://storage.googleapis.com/download.tensorflo
w.org/data/imagenet_class_index.json)
35363/35363 [==============================] - 0s 1us/step
```

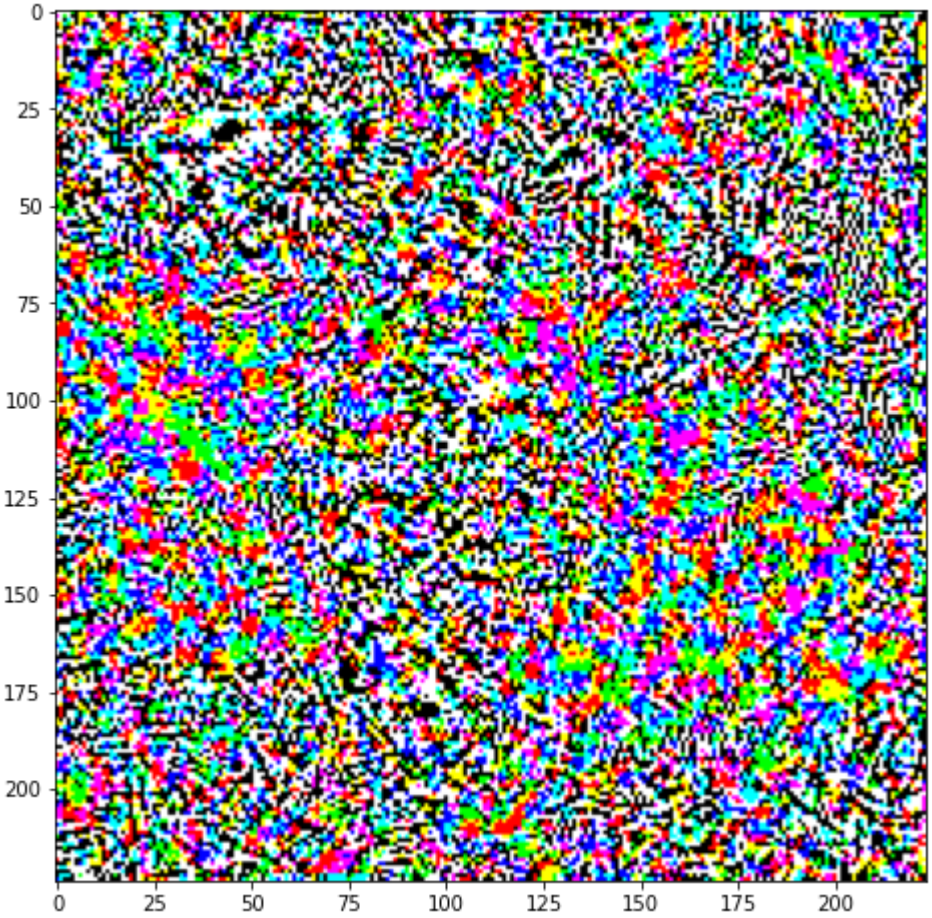
Create the Adversarial Image.

We use the FGSM method to create an adversarial image. Be sure to read about FGSM to understand how the attack works.

In [6]:

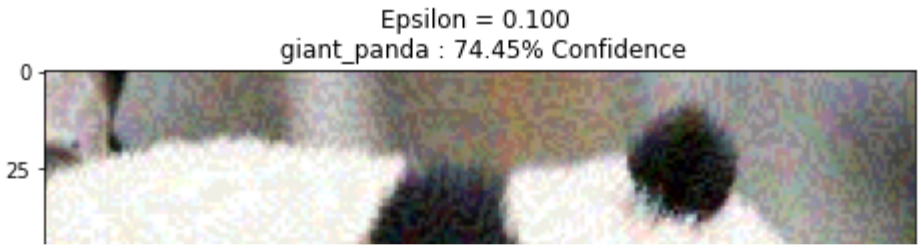
```
loss_object = tf.keras.losses.CategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = pretrained_model(input_image)
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)
  return signed_grad
```

In [7]:
```
# Get the input label of the image.
giant_panda_index = 388
label = tf.one_hot(giant_panda_index, image_probs.shape[-1])
label = tf.reshape(label, (1, image_probs.shape[-1]))

perturbations = create_adversarial_pattern(image, label)
plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1]
```



In [8]:
```
def display_images(image, description):
  _, label, confidence = get_imagenet_label(pretrained_model.predict(image))
  plt.figure()
  plt.imshow(image[0]*0.5+0.5)
  plt.title('{} \n {} : {:.2f}% Confidence'.format(description,
                                                   label, confidence*100))
  plt.show()
```

In [11]:
```
epsilons = [0, 0.01, 0.1, 0.15, 0.3]
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]
print(descriptions)
for i, eps in enumerate(epsilons):
  adv_x = image + eps*perturbations
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```



Task2: Perform an Analysis to understand the potency of the attack.

Your task here is to understand how small a change could change the class output and this is measured by the epsilon value needed to change the class.

Your task is as follows:

- Pick 10 images each from different classes in imagenet.
- Perform a perturbation analysis on each of these images.
- In the analysis you are required to report the smallest epsilon value for which you notice a class change.
- Make a table for each of the images considered with the minimum epsilon value for the FGSM attack.

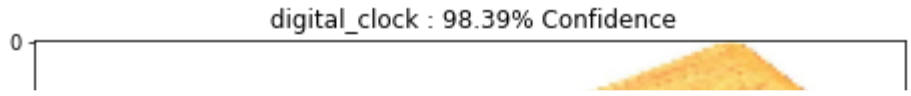
Write the Code for the above below. You can add the table also below.

10 Imagenet classes chosen

- Goldfinch 11
- Boa Constrictor 61
- Scorpion 71
- Pizza 963

- Buckle 464
- Digital clock 530
- Dutch oven 544
- Scabbard 777
- Wooden Spoon 910
- Geyser 974

In [26]:
```
# Get Images
chosen_images = []
chosen_images_probs = []
chosen_indexes = [11, 61, 71, 963, 464, 530, 544, 777, 910, 974]
for i in range(0, 10):
    image_raw = tf.io.read_file('image'+ str(i) + '.jpg')
    image = tf.image.decode_image(image_raw)
    image = preprocess(image)
    chosen_images.append(image)
    image_probs = pretrained_model.predict(image)
    chosen_images_probs.append(image_probs)

    display_images(image, "")

```



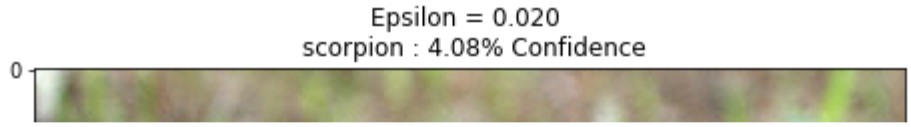
In [27]:
```
chosen_perturbations = []

for i in range(0, 10):
    label = tf.one_hot(chosen_indexes[i], chosen_images_probs[i].shape[-1])
    label = tf.reshape(label, (1, chosen_images_probs[i].shape[-1]))

    perturbations = create_adversarial_pattern(chosen_images[i], label)
#     plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1
    chosen_perturbations.append(perturbations)
```

In [28]:
```
import numpy as np
epsilons = np.arange(0, 1, 0.01)
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]
```

In [29]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[0] + eps*chosen_perturbations[0]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

In [30]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[1] + eps*chosen_perturbations[1]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

In [31]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[2] + eps*chosen_perturbations[2]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

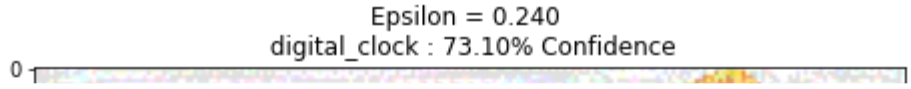


In [32]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[3] + eps*chosen_perturbations[3]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

In [33]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[4] + eps*chosen_perturbations[4]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

```
1/1 [==============================] - 0s 24ms/step
```

In [34]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[5] + eps*chosen_perturbations[5]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

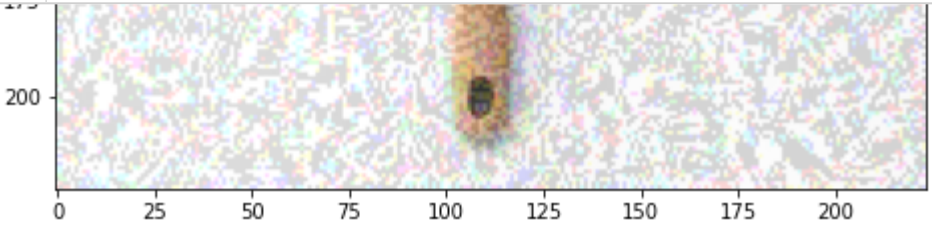


In [35]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[6] + eps*chosen_perturbations[6]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```


```
1/1 [==============================] - 0s 24ms/step
```

In [36]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[7] + eps*chosen_perturbations[7]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```

In [37]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[8] + eps*chosen_perturbations[8]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```




```
1/1 [==============================] - 0s 24ms/step
```

In [38]:
```
for i, eps in enumerate(epsilons):
  adv_x = chosen_images[9] + eps*chosen_perturbations[9]
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  display_images(adv_x, descriptions[i])
```


```
| Image Class | Minimum Epsilon for FGSM |
| ----------- | ----------- |
| Goldfinch   | 0.17 |
| Boa Constrictor | 0.01 |
| Scorpion | 0.03 |
| Pizza | 0.01 |
```

```
| Buckle | 0.01 |
| Digital clock | 0.44 |
| Dutch oven | 0.01 |
| Scabbard | 0.01 |
| Wooden Spoon | 0.28 |
| Geyser | 0.01 |
```

As one can see, the FGSM attack is very effective against MobileNet, with only a small amount of epsilon required to change an image's class most of the time. In 6/10 of the images, only an epsilon of 0.01 was required. In 3/10, a somewhat larger epsilon was required to change the class, perhaps because of the clearer backgrounds/more defining features of the object classes present, meaning the images are more far away from any decision boundaries, meaning even in the direction of greatest loss, it requires more epsilon to actually cross a decision boundary. Thus, these perturbed images are slightly more identifiable.

Task3: Compare the robustness of the considered model with other models.

Your task here is to compare how this model (MobileNetV2) compares with other popular object detection models.

Your task is as follows:

- Consider 5 different models (you can consider various RESNET architectures, any models you find interesting).
- Load the pre-trained weights of the model (trained on imagenet).
- Perform Task2 on all the considered models.
- Add all the results in the table. Hence the final table you have 6 columns for each model and epsilon values for each of the 10 images for all 6 models.

What do you observe? Why do you think this is the case?

Write the Code for the above below. You can also add the table and answer to the question below.

Models used:

```
ResNet101V2
```

```
ResNet50V2
```

```
ResNet152V2
```

```
NasNetMobile
```

```
DenseNet201
```

I had issues with VGGNet and so was not able to use it

In [59]:
```
# Helper function to preprocess the image so that it can be inputted in Mobi
def preprocess_v(image, model_ref):
  image = tf.cast(image, tf.float32)
  image = tf.image.resize(image, (224, 224))
  image = model_ref.preprocess_input(image)
  image = image[None, ...]
  return image

# Helper function to extract labels from probability vector
def get_imagenet_label_v(probs, model_ref):
    # ImageNet labels
  return model_ref.decode_predictions(probs, top=1)[0][0]
```

In [42]:
```
models = [
    tf.keras.applications.ResNet101V2(include_top=True,weights='imagenet'),
    tf.keras.applications.ResNet50V2(include_top=True,weights='imagenet'),
    tf.keras.applications.ResNet152V2(include_top=True,weights='imagenet'),
    tf.keras.applications.VGG19(include_top=True,weights='imagenet'),
    tf.keras.applications.DenseNet201(include_top=True,weights='imagenet')
]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/resnet/resnet101v2_weights_tf_dim_ordering_tf_kernels.h5 (https://storage.g
oogleapis.com/tensorflow/keras-applications/resnet/resnet101v2_weights_tf_dim_o
rdering_tf_kernels.h5)
179518384/179518384 [==============================] - 272s 2us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels.h5 (https://storage.go
ogleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ord
ering_tf_kernels.h5)
102869336/102869336 [==============================] - 50s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernels.h5 (https://storage.g
oogleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_o
rdering_tf_kernels.h5)
242745792/242745792 [==============================] - 108s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels.h5 (https://storage.googleap
is.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_ker
nels.h5)
574710816/574710816 [==============================] - 568s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/densenet/densenet201_weights_tf_dim_ordering_tf_kernels.h5 (https://storag
e.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_
dim_ordering_tf_kernels.h5)
82524592/82524592 [==============================] - 46s 1us/step
```

In [116]:
```
models[3] = tf.keras.applications.NASNetMobile(include_top=True,weights='ima
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/nasnet/NASNet-mobile.h5 (https://storage.googleapis.com/tensorflow/keras-ap
plications/nasnet/NASNet-mobile.h5)
24227760/24227760 [==============================] - 12s 1us/step
```

In [117]:
```
model_refs = [
    tf.keras.applications.resnet_v2,
    tf.keras.applications.resnet_v2,
    tf.keras.applications.resnet_v2,
    tf.keras.applications.nasnet,
    tf.keras.applications.densenet
]
```

In [118]:
```
for i in range(0, 5):
    models[i].trainable = False
```

In [119]:
```
loss_object = tf.keras.losses.CategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label, model):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = model(input_image)
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)
  return signed_grad
```

In [120]:
```
# print(tf.keras.applications.resnet101v2.decode_predictions)
```

In [121]:

```
model_images = []
model_images_probs = []
chosen_indexes = [11, 61, 71, 963, 464, 530, 544, 777, 910, 974]
for j in range(0, 5):
    model = models[j]
    model_ref = model_refs[j]
    images = []
    probs = []
    for i in range(0, 10):
        image_raw = tf.io.read_file('image'+ str(i) + '.jpg')
        image = tf.image.decode_image(image_raw)
        image = preprocess_v(image, model_ref)
        images.append(image)
        image_probs = model.predict(image)
        probs.append(image_probs)

#         display_images(image, "")
    model_images.append(images)
    model_images_probs.append(probs)

```

```
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
```

```
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 100ms/step
```

In [122]:

```
1
2  model_perturbations = []
3  for j in range(0, 5):
4      perturbs = []
5      for i in range(0, 10):
6          label = tf.one_hot(chosen_indexes[i], model_images_probs[j][i].shape
7          label = tf.reshape(label, (1, model_images_probs[j][i].shape[-1]))
8
9          perturbations = create_adversarial_pattern(model_images[j][i], label
10     #     plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to
11         perturbs.append(perturbations)
12     model_perturbations.append(perturbs)
```

In [126]:

```
im_labels = [
"goldfinch",
"boa_constrictor",
"scorpion",
"pizza",
"buckle",
"digital_clock",
"Dutch_oven",
"scabbard",
"wooden_spoon",
"geyser"
]

def get_lowest_epsilon_and_display(image, pert, actual_label, model, model_r
  eps = 0
  adv_x = image + eps*pert
  adv_x = tf.clip_by_value(adv_x, -1, 1)
  _, label, confidence = get_imagenet_label_v(model.predict(adv_x), model_re
  while label == actual_label:
      eps = eps + 0.01
      adv_x = image + eps*pert
      adv_x = tf.clip_by_value(adv_x, -1, 1)
      _, label, confidence = get_imagenet_label_v(model.predict(adv_x), mode
#       print(eps)
#       print(label)
#       print(actual_label)
  plt.figure()
  plt.imshow(adv_x[0]*0.5+0.5)
  plt.title('Epsilon = {:0.3f}\n{} : {:.2f}% Confidence'.format(eps, label,
  plt.show()
  return eps
```

In [127]:
```
j = 0
model_epsilons = []
epsilons = []
for i in range(0, 10):
    epsilons.append(get_lowest_epsilon_and_display(model_images[j][i], model
model_epsilons.append(epsilons)
```

```
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 84ms/step
```

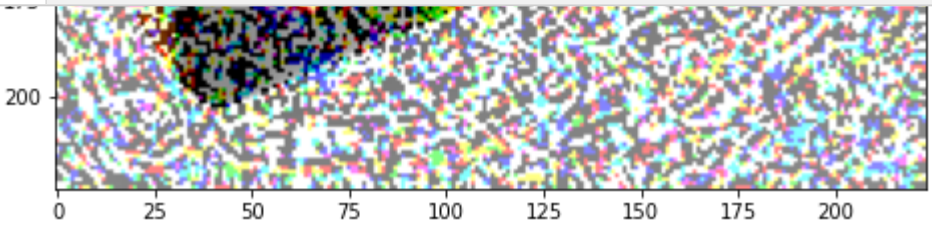
In [128]:
```
print(model_epsilons)
```

```
[[0.35000000000000014, 0.03, 0.02, 0.01, 0.6600000000000004, 1.020000000000000
7, 0.01, 0.02, 0.4100000000000002, 0.01]]
```

In [129]:
```
j = 1
epsilons = []
for i in range(0, 10):
    epsilons.append(get_lowest_epsilon_and_display(model_images[j][i], model
model_epsilons.append(epsilons)
```

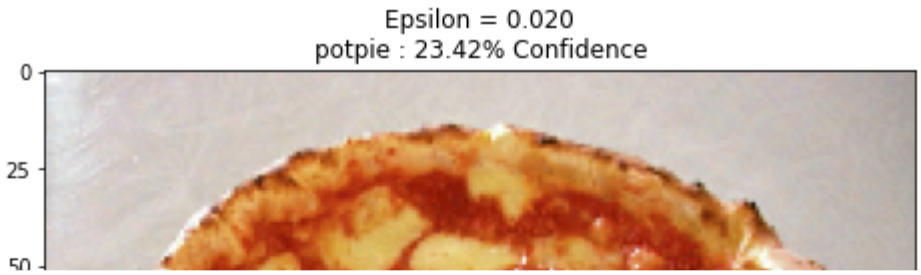
```
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 57ms/step
```

In [131]:
```
j = 2
epsilons = []
for i in range(0, 10):
    epsilons.append(get_lowest_epsilon_and_display(model_images[j][i], model
model_epsilons.append(epsilons)
```




```
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 142ms/step
```

In [132]:
```
j = 3
epsilons = []
for i in range(0, 10):
    epsilons.append(get_lowest_epsilon_and_display(model_images[j][i], model
model_epsilons.append(epsilons)
```



In [133]:
```
= []
range(0, 10):
ons.append(get_lowest_epsilon_and_display(model_images[j][i], model_perturbations
ilons.append(epsilons)
```

In [135]:
```
print(model_epsilons)
```

```
[[0.35000000000000014, 0.03, 0.02, 0.01, 0.6600000000000004, 1.020000000000000
7, 0.01, 0.02, 0.4100000000000002, 0.01], [0.2700000000000001, 0.03, 0.2700000
00000001, 0.01, 0.4200000000000002, 0.03, 0.01, 0.01, 0.3900000000000002, 0.0
1], [0.38000000000000017, 0.13999999999999999, 0.3300000000000001, 0.01, 0.4800
0000000000026, 0.9100000000000006, 0.01, 0.01, 0.4000000000000002, 0.01], [0.27
00000000000001, 0.01, 0.34000000000000014, 0.02, 0.7300000000000004, 0.96000000
00000006, 0.01, 0.01, 0.38000000000000017, 0.05], [0.5900000000000003, 0.07, 0.
060000000000000005, 0.04, 0.35000000000000014, 1.2300000000000009, 0.15, 0.04,
0.10999999999999999, 0.02]]
```

In [139]:
```
mobileneteps = [0.17,0.01,0.03,0.01,0.01,0.44,0.01,0.01,0.28,0.01]
print(np.mean(model_epsilons, axis=1))
print(np.mean(mobileneteps))

```

```
[0.254 0.145 0.268 0.278 0.266]
0.098
```

In general, the ResNet, NASNet and DenseNet models were more robust to the FGSM attack than MobileNet. This is shown by the fact that the average required epsilon to change the class of an image was higher for all the tested models compared to MobileNet. This is probably the case because of the higher complexity of the other models as compared to MobileNet. As one can see in the code above, the other models have a lot more weights (on the order of 10^8 as opposed to MobileNet's approximately 10^4 weights). Thus, because the models have more complex functions, their loss functions are also more complex, making it harder to find and navigate to a decision boundary for a given image.

Image Class	MobileNet	ResNet101V2	ResNet50V2	ResNet152V2	NasNetMobile	DenseNet201
Goldfinch	0.17	0.35	0.27	0.38	0.27	0.59
Boa Constrictor	0.01	0.03	0.03	0.14	0.01	0.07
Scorpion	0.03	0.02	0.27	0.33	0.34	0.06
Pizza	0.01	0.01	0.01	0.01	0.02	0.04
Buckle	0.01	0.66	0.42	0.48	0.73	0.35
Digital clock	0.44	1.02	0.03	0.91	0.96	1.23
Dutch oven	0.01	0.01	0.01	0.01	0.01	0.15
Scabbard	0.01	0.02	0.01	0.01	0.01	0.04
Wooden Spoon	0.28	0.41	0.39	0.40	0.38	0.11
Geyser	0.01	0.01	0.01	0.01	0.05	0.02

BONUS: Can you provide a better attack?

Can you design a better attack that lowers the epsilon required for the images?

Task:

- Design another attack.
- Compare the epsilon values on 10 images.
- Does it perform better than the FGSM attack? That is, does it have lower epsilon values?

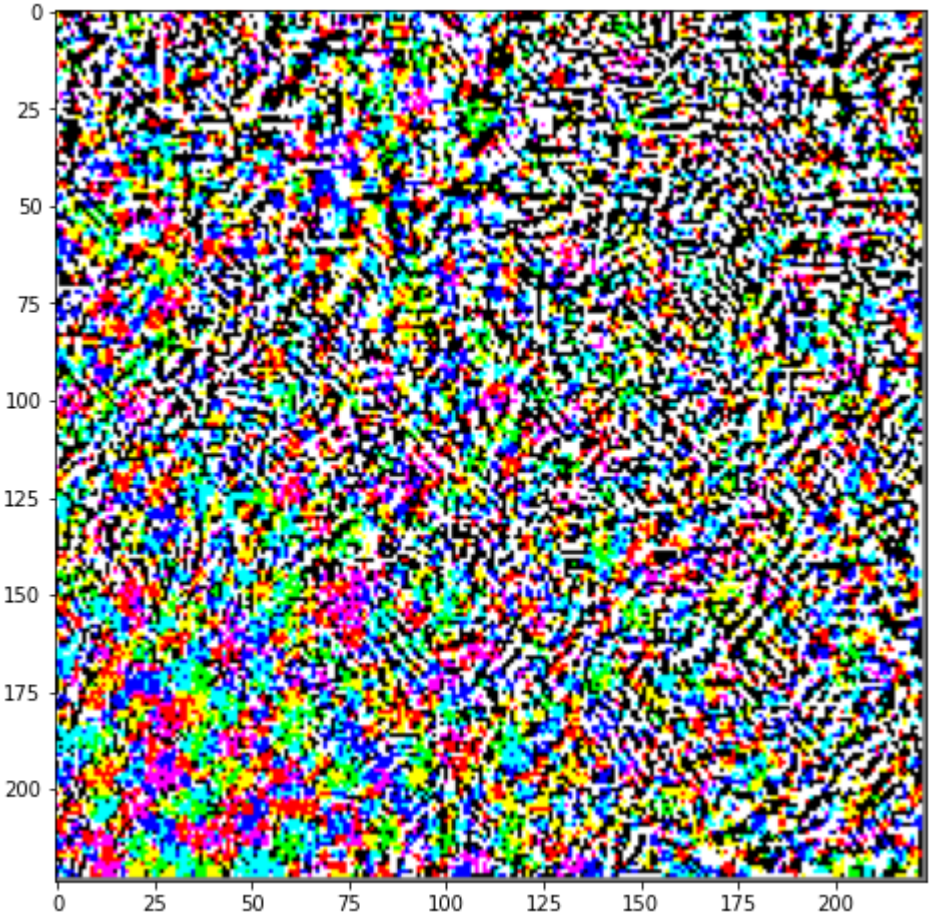
Write the code and provide your answers below.

In [179]:
```
def create_adversarial_another(input_image, input_label, model):
  #Add random noise to the input image (RAND-FGSM)

  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = model(input_image + 0.001 * np.random.normal(size=(1, 224,
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)
  return signed_grad
```

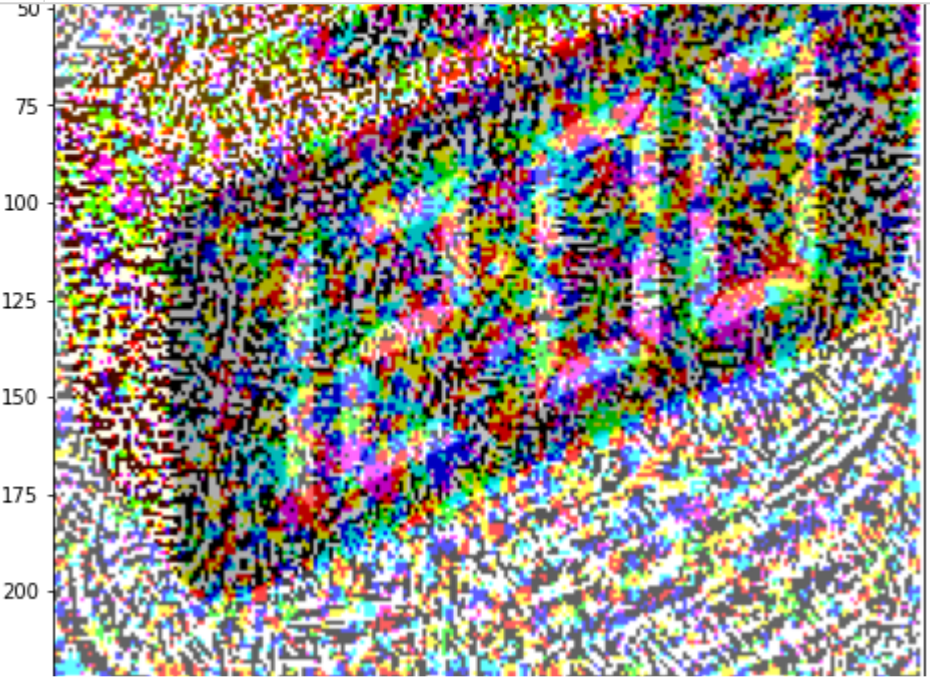
In [180]:
```
probs = []
images = []
model = models[3]
model_ref = model_refs[3]

for i in range(0, 10):
    image_raw = tf.io.read_file('image'+ str(i) + '.jpg')
    image = tf.image.decode_image(image_raw)
    image = preprocess_v(image, model_ref)
    images.append(image)
    image_probs = model.predict(image)
    probs.append(image_probs)

#         display_images(image, "")
    model_images.append(images)
    model_images_probs.append(probs)
```

```
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
```

In [181]:
```
perbs = []

for i in range(0, 10):
    label = tf.one_hot(chosen_indexes[i], probs[i].shape[-1])
    label = tf.reshape(label, (1, probs[i].shape[-1]))

    perturbations = create_adversarial_another(images[i], label, model)
    plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1]
    perbs.append(perturbations)
```



In [182]:
```
epsilons = []
for i in range(0, 10):
    epsilons.append(get_lowest_epsilon_and_display(images[i], perbs[i], im_l
```



In [183]:
```
print(epsilons)
print(np.mean(epsilons))
```

```
[0.22000000000000006, 0.01, 0.3100000000000001, 0.02, 0.7000000000000004, 1.180
0000000000008, 0.01, 0.01, 0.37000000000000016, 0.060000000000000005]
0.28900000000000015
```

I attempted to add random variation when updating the adversarial samples-- this is to help mask the gradient and overcome any adversarial training the model might have. We end up with worse performance, as the required epsilons are overall higher. This is because while adding the random mask might help with any existing robustness measures, it actively harms our attempts to get to the direction of the optimal loss to get to the nearest decision boundary. Thus, it takes more epsilon to actually affect the image.