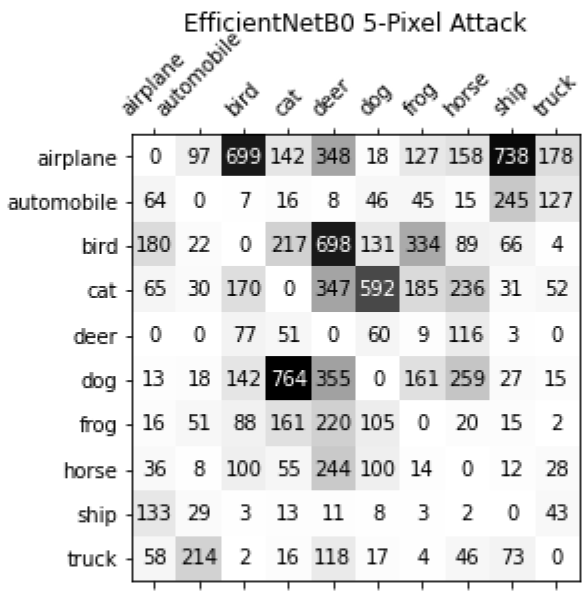
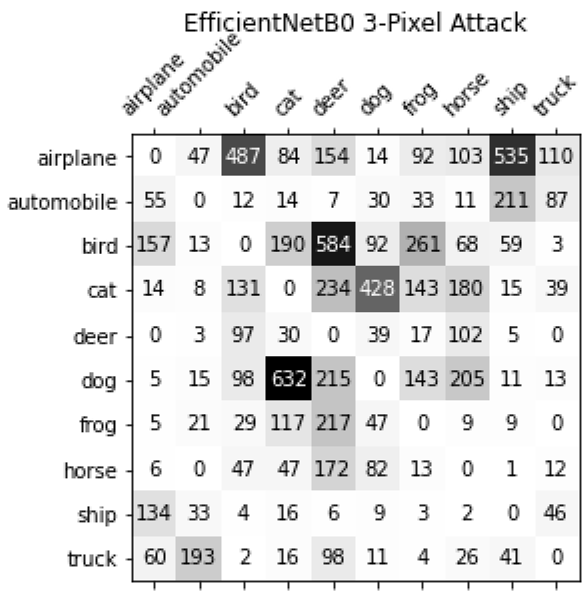
Dong Min Shin

Replication of Heatmaps in “One Pixel Attack for Fooling Deep Neural Networks”

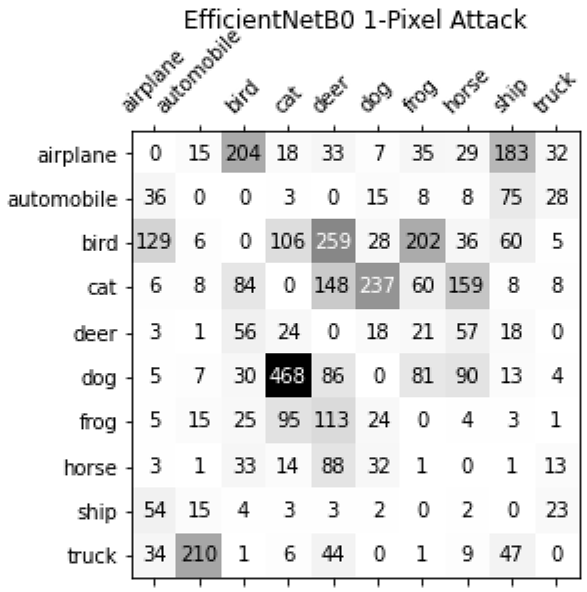
The original paper describes how image classifier models can produce a wrong answer, given only a change in a single pixel through differential evolution. DE is a non-gradient descent based algorithm which generates child solutions when given parent solutions from the previous iteration. The algorithm in the paper seeks to maximize the probability of an adversarial image f_adv(x + e) where x has dimensions equal to the number of pixels, and the L0-norm of the error should be 1. In other words, the error should be confined to a single pixel without any regard to the difference in the RGB value at that pixel.


EfficientNetB0 5-Pixel Attack

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0	97	699	142	348	18	127	158	738	178
automobile	64	0	7	16	8	46	45	15	245	127
bird	180	22	0	217	698	131	334	89	66	4
cat	65	30	170	0	347	592	185	236	31	52
deer	0	0	77	51	0	60	9	116	3	0
dog	13	18	142	764	355	0	161	259	27	15
frog	16	51	88	161	220	105	0	20	15	2
horse	36	8	100	55	244	100	14	0	12	28
ship	133	29	3	13	11	8	3	2	0	43
truck	58	214	2	16	118	17	4	46	73	0


EfficientNetB0 3-Pixel Attack

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0	47	487	84	154	14	92	103	535	110
automobile	55	0	12	14	7	30	33	11	211	87
bird	157	13	0	190	584	92	261	68	59	3
cat	14	8	131	0	234	428	143	180	15	39
deer	0	3	97	30	0	39	17	102	5	0
dog	5	15	98	632	215	0	143	205	11	13
frog	5	21	29	117	217	47	0	9	9	0
horse	6	0	47	47	172	82	13	0	1	12
ship	134	33	4	16	6	9	3	2	0	46
truck	60	193	2	16	98	11	4	26	41	0

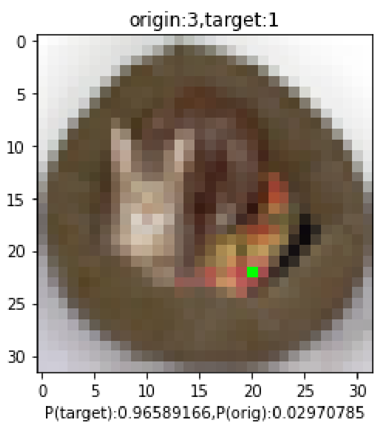

EfficientNetB0 1-Pixel Attack

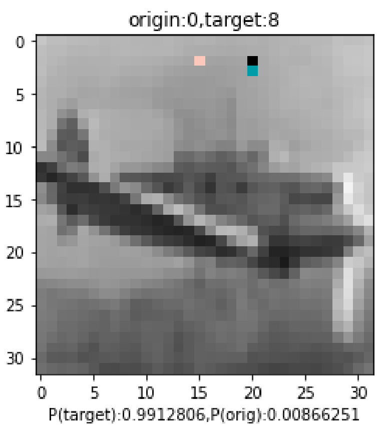
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0	15	204	18	33	7	35	29	183	32
automobile	36	0	0	3	0	15	8	8	75	28
bird	129	6	0	106	259	28	202	36	60	5
cat	6	8	84	0	148	237	60	159	8	8
deer	3	1	56	24	0	18	21	57	18	0
dog	5	7	30	468	86	0	81	90	13	4
frog	5	15	25	95	113	24	0	4	3	1
horse	3	1	33	14	88	32	1	0	1	13
ship	54	15	4	3	3	2	0	2	0	23
truck	34	210	1	6	44	0	1	9	47	0

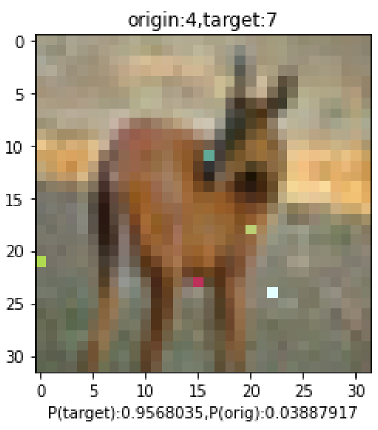
Figure 6 is a series of heatmaps describing the number of successful 1-,3-,5-pixel attacks from a given original label to 9 other labels. While the figure uses VGG16 and AllConv networks, Keras EfficientNetB0 + transfer learning model was used for the replication, as the prebuilt Keras VGG16 has a large parameter size which makes it cumbersome for transfer learning, and AllConv did not have any pre trained models. Since the model takes in 224x224x3 shape as an input, the 32x32x3 CIFAR images are first upsampled by factor of 7, then two fully connected size 1024 layers each with 0.2 dropout after are added at the end, then a size 10 layer with softmax as the output.

The original paper separately computes the number of successful attacks from the origin to target label by running the DE algorithm 9 times for each origin label. The problem is that each evaluation of the algorithm requires its own iterations of adversarial image generation and image classifier model output as the fitness function. This becomes very costly in both memory and time, so instead an untargeted attack is made for each origin label, which the paper actually uses for the attacks on ImageNet model for the same reason of saving memory and time costs.

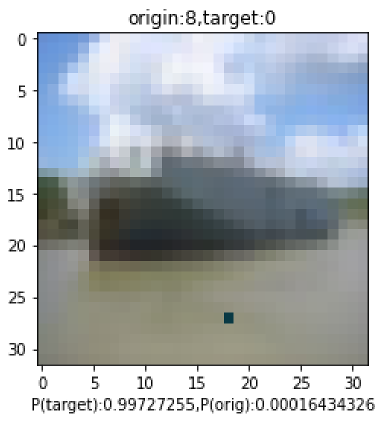
In general, the heatmap seems to be a bit less symmetric than the figure on the paper. Considering that the transferred EfficientNetB0 only had an accuracy value of about 80%, a further time spent training the model, or even data augmentation could have made the network more resilient against more diverse types of noise, which could have made the heatmap more symmetric as well. Similar to what the paper shows, the increase in the number of pixels also increases the total number of attacks, but does not change the structure of the heatmap.

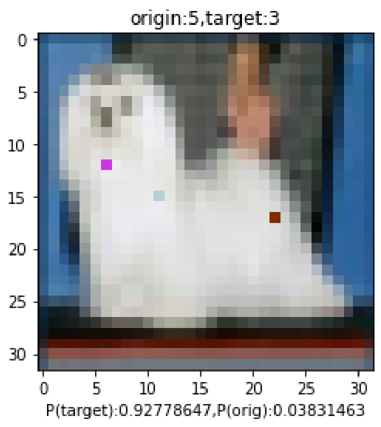


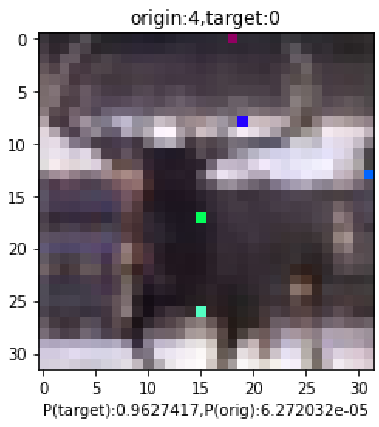




Dong Min Shin







J. Su, D.V. Vargas, K. Sakurai, One Pixel Attack for Fooling Deep Neural Networks, [1710.08864.pdf (arxiv.org)](1710.08864.pdf)

```
def diff_evo_iterations(inputs, classes, image, model, preprocess, ans, target,
iterations=20,pixels=1):
    batch_size = inputs.shape[0]
    f = lambda start_idx: tf.random.shuffle(np.r_[(start_idx+1):batch_size,
:start_idx].astype('int32'))
    minclip = tf.tile([[0,0,0,0,0]],[pixels,1])
    maxclip = tf.tile([[31,31,255,255,255]],[pixels,1])
    prevImg = None
    y_pred = None
    for j in range(iterations):
        diag_less = tf.map_fn(f,tf.cast(tf.range(batch_size),tf.int32))
        r1,r2,r3 =
tf.gather(inputs.astype('int32'),tf.transpose(diag_less[:,0:3]),axis=0)
        i = tf.clip_by_value(r1 + (r2 - r3)//2,
                             minclip,maxclip)
        img = np.zeros([batch_size,32,32,3],dtype='uint8')
        for (num,params) in enumerate(i):
            img[num] = image.copy()
            for par in params:
                img[num,par[0],par[1]] = tf.cast(par[2:5],dtype=tf.uint8).numpy()
        if prevImg is None:
            y_pred = model.predict(preprocess(img),batch_size=64)
            prevImg = img
        else:
            y_tmp = tf.concat([y_pred,model.predict(preprocess(img),batch_size=64)],0)
            if target is None:
              indices = tf.argsort(y_tmp[:,ans],direction='ASCENDING')[:batch_size]
            else:
              indices =
tf.argsort(y_tmp[:,target],direction='DESCENDING')[:batch_size]
```

```
            prevImg = tf.gather(tf.concat([prevImg,img],0),indices)
            y_pred = tf.gather(y_tmp,indices)
            if (target is not None and y_pred[0][target] >= 0.9):
                break
            elif target is None and y_pred[0][ans] <= 0.05:
                break
    if target is None:
        return [[prevImg[i],y_pred[i][arg],y_pred[i][ans],arg.numpy()] for (i,arg) in
enumerate(tf.argmax(y_pred,1)) if arg.numpy() != ans]
    else:
        return [[prevImg[i],y_pred[i][arg],y_pred[i][ans]] for (i,arg) in
enumerate(tf.argmax(y_pred,1)) if arg == target]

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
def diff_evo_init(batch_size=64,model=None,preprocess=None,X=None,Y=None,pixels=1):
    i =
np.concatenate((np.random.uniform(0,31,(batch_size,pixels,2)).astype('int32'),np.clip(
np.random.normal(128,127,(batch_size,pixels,3)).astype('int32'),0,255)),axis=2,dtype='
int32')
    l = []
    y_preds = model.predict(preprocess(X),batch_size=128,verbose=1)
    y_preds
    bs = np.equal(Y[:,0], np.argmax(y_preds, axis=1))
    print(Y[:,0][:36],np.argmax(y_preds, axis=1)[:36],y_preds[:36])
    l = np.asarray(bs==True).nonzero()[0]
    ll=[[] for i in range(10)]
    for q in l:
        ll[Y[q,0]] += [q]
    for ans in range(10):
        print('original label: ',ans)
        dump = []
        for ind,imgind in enumerate(ll[ans][:50]):
            print('index, img index: ',ind,imgind)
            res =
diff_evo_iterations(i,class_names,X[imgind],model,preprocess,ans,None,pixels=pixels)
            print(len(res))
            if len(res) > 0:
                dump += [r[:-1] + [ans,ind,imgind,r[-1]] for r in res]
        with open('/content/drive/MyDrive/diffevo'+str(pixels)+str(ans), 'wb') as
file:
            pickle.dump(dump, file)
diff_evo_init(model=models[0],preprocess=efficientnet.preprocess_input,X=X,Y=Y,pixels=
5)

def load_results(path,pix=1):
  results = []
  newold=[pix]*10
  for i in [0,1,2,3,4,5,6,7,8,9]:
    if newold[i]>0:
      s = str(newold[i])
    else:
      s = ''
    with open(os.path.join(path,'diffevo'+s+str(i)),'rb') as file:
      load = pickle.load(file)
      results += load
  return results

def origTargetMatrix(results,title):
  matrix = np.zeros([10,10])
  for table in results:
    matrix[table[3],table[-1]] += 1
  im=plt.matshow(np.transpose(matrix),cmap='Greys')
  textcolors=['k','w']
  plt.title(title,pad=40)
  plt.xticks(np.arange(0,10),class_names,rotation=45)
  plt.yticks(np.arange(0,10),class_names)
  for i in range(10):
    for j in range(10):
      text = plt.gca().text(j, i, int(matrix[j,i]),
                        ha="center", va="center",
color=textcolors[int(im.norm(matrix[j,i]) > 0.5)])
res1 = load_results('/content/drive/MyDrive',1)
res3 = load_results('/content/drive/MyDrive',3)
res5 = load_results('/content/drive/MyDrive',5)
origTargetMatrix(res1,'EfficientNetB0 1-Pixel Attack')
origTargetMatrix(res3,'EfficientNetB0 3-Pixel Attack')
origTargetMatrix(res5,'EfficientNetB0 5-Pixel Attack')
for res in [res1, res3, res5]:
  l = []
  for table in res:
    if table[1] > 0.9 and table[2] < 0.05:
      l.append(table)
  i = np.random.uniform(0,len(l),(10,)).astype('int32')
  print(len(l))
  for t in i:
    t = l[t]
    plt.figure()
    plt.imshow(t[0])
    plt.title('origin:'+str(t[3])+',target:'+str(t[-1]))
    plt.xlabel('P(target):'+str(t[1].numpy())+',P(orig):'+str(t[2].numpy()))
```

Dong Min Shin