Name: Hussein Abdallah

ID: 40185921

1.A

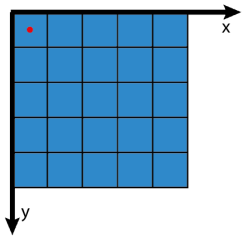
https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature

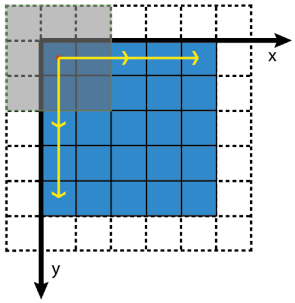
k : convolution kernel size
p : convolution padding size
s : convolution stride size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k-1) * j_{in}$$
$$start_{out} = start_{in} + \left(\frac{k-1}{2} - p\right) * j_{in}$$

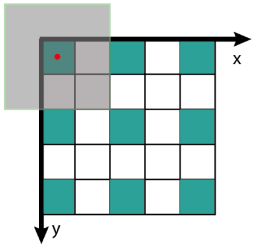
Layer 0: $n_0 = 5;\ r_0 = 1;\ j_0 = 1;\ start_0 = 0.5$



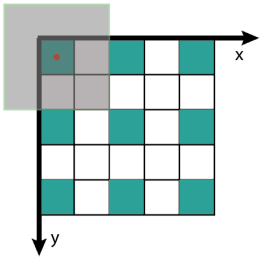
Conv1: $k_1 = 3;\ p_1 = 1;\ s_1 = 2$



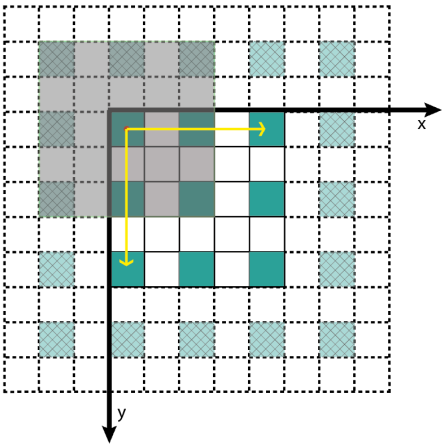
Layer 1: $n_1 = 3;\ r_1 = 3;\ j_1 = 2;\ start_1 = 0.5$



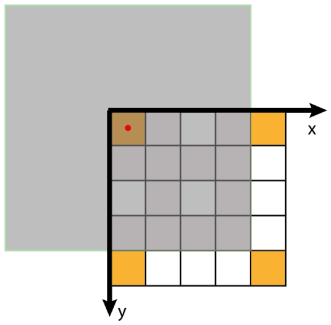
Layer 1: $n_1 = 3;\ r_1 = 3; j_1 = 2;\ start_1 = 0.5$



Conv2: $k_2 = 3;\ p_2 = 1;\ s_2 = 2$



Layer 2: $n_3 = 2;\ r_3 = 7; j_3 = 4;\ start_1 = 0.5$



In [11]:

```
# [filter size, stride, padding]
#Assume the two dimensions are the same
#Each kernel requires the following parameters:
# - k_i: kernel size
# - s_i: stride
# - p_i: padding (if padding is uneven, right padding will higher than left padding; "S
#
#Each layer i requires the following parameters to be fully represented:
# - n_i: number of feature (data layer has n_1 = imagesize )
# - j_i: distance (projected to image pixel distance) between center of two adjacent fe
# - r_i: receptive field of a feature in layer i
```

```
# - start_i: position of the first feature's receptive field in layer i (idx start from

import math
convnet =   [[11,4,0],[3,2,0],[5,1,2],[3,2,0],[3,1,1],[3,1,1],[3,1,1],[3,2,0],[6,1,0],
layer_names = ['conv1','pool1','conv2','pool2','conv3','conv4','conv5','pool5','fc6-con
imsize = 227

def outFromIn(conv, layerIn):
    n_in = layerIn[0]
    j_in = layerIn[1]
    r_in = layerIn[2]
    start_in = layerIn[3]
    k = conv[0]
    s = conv[1]
    p = conv[2]

    n_out = math.floor((n_in - k + 2*p)/s) + 1
    actualP = (n_out-1)*s - n_in + k
    pR = math.ceil(actualP/2)
    pL = math.floor(actualP/2)

    j_out = j_in * s
    r_out = r_in + (k - 1)*j_in
    start_out = start_in + ((k-1)/2 - pL)*j_in
    return n_out, j_out, r_out, start_out

def printLayer(layer, layer_name):
    print(layer_name + ":")
    print("\t n features: %s \n \t jump: %s \n \t receptive size: %s \t start: %s " % (

layerInfos = []
#first layer is the data layer (image) with n_0 = image size; j_0 = 1; r_0 = 1; and sta
# print ("-------Net summary------")
currentLayer = [imsize, 1, 1, 1]
printLayer(currentLayer, "input image")

for i in range(len(convnet)):
    currentLayer = outFromIn(convnet[i], currentLayer)
    layerInfos.append(currentLayer)
    printLayer(currentLayer, layer_names[i])
print ("-------------------------")
layer_name ="conv1"
print_layer_names = ['conv1','pool1','conv2','pool2']
for layer_name in  print_layer_names:
    layer_idx = layer_names.index(layer_name)
    idx_x = 1
    idx_y = 1
    n = layerInfos[layer_idx][0]
    j = layerInfos[layer_idx][1]
    r = layerInfos[layer_idx][2]
    start = layerInfos[layer_idx][3]
    if(idx_x < n):
#     assert(idx_y < n)
        print(layer_name,':')
        print('number of features=',n)
        print ("receptive field: (%s, %s)" % (r, r))
        print ("start: (%s, %s)" % ((start+idx_x*j)-r/2, (start+idx_y*j)-r/2))
        print ("center: (%s, %s)" % (start+idx_x*j, start+idx_y*j))
        print ("end: (%s, %s)" % ((start+idx_x*j)+r/2, (start+idx_y*j)+r/2))
        print('-------------------------')
```

```
input image:
         n features: 227
         jump: 1
         receptive size: 1       start: 1
conv1:
         n features: 55
         jump: 4
         receptive size: 11      start: 6.0
pool1:
         n features: 27
         jump: 8
         receptive size: 19      start: 10.0
conv2:
         n features: 27
         jump: 8
         receptive size: 51      start: 10.0
pool2:
         n features: 13
         jump: 16
         receptive size: 67      start: 18.0
conv3:
         n features: 13
         jump: 16
         receptive size: 99      start: 18.0
conv4:
         n features: 13
         jump: 16
         receptive size: 131     start: 18.0
conv5:
         n features: 13
         jump: 16
         receptive size: 163     start: 18.0
pool5:
         n features: 6
         jump: 32
         receptive size: 195     start: 34.0
fc6-conv:
         n features: 1
         jump: 32
         receptive size: 355     start: 114.0
fc7-conv:
         n features: 1
         jump: 32
         receptive size: 355     start: 114.0
------------------------
conv1 :
number of features= 55
receptive field: (11, 11)
start: (4.5, 4.5)
center: (10.0, 10.0)
end: (15.5, 15.5)
------------------------
pool1 :
number of features= 27
receptive field: (19, 19)
start: (8.5, 8.5)
center: (18.0, 18.0)
end: (27.5, 27.5)
------------------------
conv2 :
number of features= 27
receptive field: (51, 51)
start: (-7.5, -7.5)
center: (18.0, 18.0)
end: (43.5, 43.5)
```

```
-------------------------
pool2 :
number of features= 13
receptive field: (67, 67)
start: (0.5, 0.5)
center: (34.0, 34.0)
end: (67.5, 67.5)
-------------------------
```

1.B

In [14]:
```
import torch.nn as nn
import numpy as np
import torch as torch
import cv2
from torch.autograd import Variable
import torch.nn.functional as F
from torch.optim import Adam
model = torch.hub.load('pytorch/vision:v0.9.0', 'alexnet', pretrained=True)
model.eval()
```

```
Using cache found in C:\Users\Administrator/.cache\torch\hub\pytorch_vision_v0.9.0
```

Out[14]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In [15]:
```
# def get_img_tensor(img_file_name):
#     input_image = Image.open(img_file_name)
# #     print(input_image)
#     preprocess = transforms.Compose([
#         transforms.Resize(224),
#         transforms.CenterCrop(224),
#         transforms.ToTensor(),
#         transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
#     ])
#     return preprocess(input_image)
```

In [16]:
```
def preprocess(image_path):
    ALEX_MEAN = [0.485, 0.456, 0.406]
    ALEX_STD=[0.229, 0.224, 0.225]
    original_image = cv2.imread(image_path)
    image = np.copy(original_image)

    out = cv2.resize(image, (224, 224))
    out = out.astype(np.float32) / 256.
    image = np.clip(image, 0, 1)
    out = out[:, :, [2, 1, 0]]  # swap channel from BGR to RGB
    out[:, :, 0] = (out[:, :, 0] - ALEX_MEAN[0]) / ALEX_STD[0]
    out[:, :, 1] = (out[:, :, 1] - ALEX_MEAN[1]) / ALEX_STD[1]
    out[:, :, 2] = (out[:, :, 2] - ALEX_MEAN[2]) / ALEX_STD[2]
    out = np.transpose(out, (2, 0, 1))
    return out
```

In [17]:
```
def deprocess(tensor):
    ALEX_MEAN = [0.485, 0.456, 0.406]
    ALEX_STD=[0.229, 0.224, 0.225]

    out = tensor.data.cpu().numpy()
    out = np.reshape(out,[3, 224, 224])
    out = np.transpose(out, (1, 2, 0))

    out[:, :, 0] = (out[:, :, 0] * ALEX_STD[0]) + ALEX_MEAN[0]
    out[:, :, 1] = (out[:, :, 1] * ALEX_STD[1]) + ALEX_MEAN[1]
    out[:, :, 2] = (out[:, :, 2] * ALEX_STD[2]) + ALEX_MEAN[2]
    out = out[:, :, [2, 1, 0]]  # swap channel from RGB to BGR
    out = np.clip(out, 0, 1)
    out = out * 256

    out = out.astype(np.uint8)
    return out
```

In [30]:
```
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
def get_image_label(img, is_tensor=False):
    if is_tensor ==False:
        input_tensor =torch.from_numpy(preprocess(img))
    else:
        input_tensor=img
    input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the mo

    # move the input and model to GPU for speed if available
    if torch.cuda.is_available():
        input_batch = input_batch.to('cuda')
        model.to('cuda')

    with torch.no_grad():
        output = model(input_batch)
    # Tensor of shape 1000, with confidence scores over Imagenet's 1000 classes
    # print(output[0])
    # The output has unnormalized scores. To get probabilities, you can run a softmax o
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    # print(probabilities)
```

```
    # Read the categories
    with open("imagenet_classes.txt", "r") as f:
        categories = [s.strip() for s in f.readlines()]
    # Show top categories per image
    # top5_prob, top5_catid = torch.topk(probabilities, 5)
    # for i in range(top5_prob.size(0)):
    #     print(categories[top5_catid[i]], top5_prob[i].item())

    top_prob, top_catid = torch.topk(probabilities, 1)
#     print(top_prob)
    return top_catid[0],categories[top_catid[0]]
```

In [62]:

```
from imageio import imread
import matplotlib.pyplot as plt
urls=[["https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n01726692_4
      ["https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n01674464_3
      ["https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n01770393_1
      ["https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n02129165_2
      ["https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n02129604_2

for url in urls:
#     image = imread(url[0])
#     plt.imshow(image)
    # url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog
    # !wget("https://github.com/ajschumacher/imagen/blob/master/imagen/n00007846_147031_
    http_url, filename = (url[0], url[1])
    label_code,label= get_image_label(filename)
    url.append(label_code.item())
    print("ground truth=",filename," predicted=",label)
```

```
ground truth= snake.jpg  predicted= water snake
ground truth= lizard.jpg  predicted= agama
ground truth= scorpion.jpg  predicted= scorpion
ground truth= lion.jpg  predicted= lion
ground truth= tiger.jpg  predicted= tiger
```

In [64]:

```
print(urls)
```

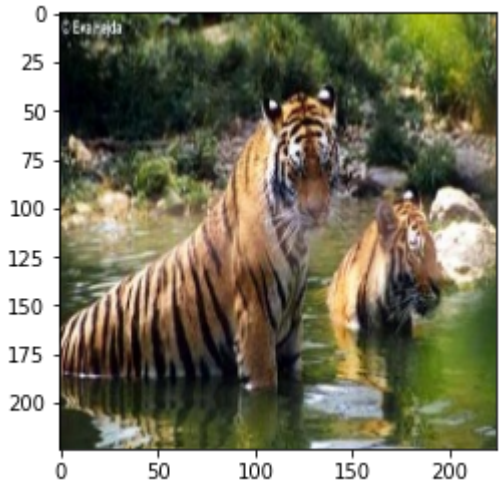
```
[['https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n01726692_4802_sn
ake.jpg', 'snake.jpg', 58], ['https://raw.githubusercontent.com/ajschumacher/imagen/mast
er/imagen/n01674464_3490_lizard.jpg', 'lizard.jpg', 42], ['https://raw.githubuserconten
t.com/ajschumacher/imagen/master/imagen/n01770393_12410_scorpion.jpg', 'scorpion.jpg', 7
1], ['https://raw.githubusercontent.com/ajschumacher/imagen/master/imagen/n02129165_2762
_lion.jpg', 'lion.jpg', 291], ['https://raw.githubusercontent.com/ajschumacher/imagen/ma
ster/imagen/n02129604_20374_tiger.jpg', 'tiger.jpg', 292]]
```

1.C

In [34]:

```
img = preprocess(urls[4][1])
img_tensor = torch.from_numpy(img)
img_tensor = img_tensor.unsqueeze_(0)
img_variable = Variable(img_tensor, requires_grad=True)

pre_image = deprocess(img_tensor)
cv2.imwrite("pre_processed.jpg", pre_image)
plt.figure()
plt.imshow(cv2.cvtColor(cv2.imread("pre_processed.jpg"),cv2.COLOR_BGR2RGB))
get_image_label("pre_processed.jpg")
```

Out[34]: (tensor(292), 'tiger')



In [75]:
```
# def generate_adverserial(input_image,ground_truth):
#     print('target_variable=',target_variable)
#     optimizer = Adam([img_variable], lr=0.005)
#     criteron = nn.CrossEntropyLoss()
#     print("\nStarting Optimization...")
#     for iteration in range(1,100):
#         final_image = deprocess(img_variable)
#         cv2.imwrite("temp.jpg", final_image)
#         adv_label=get_image_label("temp.jpg")[0].item()
#         print('adv_label',adv_label)
#         if adv_label==ground_truth:
#             fc_out = model(input_image)
#             fc_out = F.softmax(fc_out, dim=1)

#             optimizer.zero_grad()
#             loss = criteron(fc_out, target_variable)
#             print(img_variable.grad.shape)

#             loss.backward(retain_graph = True)
#         #     grads = torch.sign(img_variable.grad.data)
#         #     print(img_variable.grad.data)
#         #     optimizer.zero_grad()
#             optimizer.step()
#         #     epsilon = [0.0001, 0.001, 0.0013, 0.009, 0.015, 0.120, 0.25, 0.302]
#         #     for i in epsilon:
#         #         img_variable = img_variable.data + i

#             if iteration == 1 or iteration%50 == 0:
#                 print("Current Iteration: ",iteration,"\tCurrent Cost: ", loss)

#         else:
#             break
#     return input_image
```

In [77]:
```
def generate_adverserial(input_image,target_variable,iterations):
    optimizer = Adam([img_variable], lr=0.005)
    criteron = nn.CrossEntropyLoss()
    print("\nStarting Optimization...")
    for iteration in range(1,iterations):
```

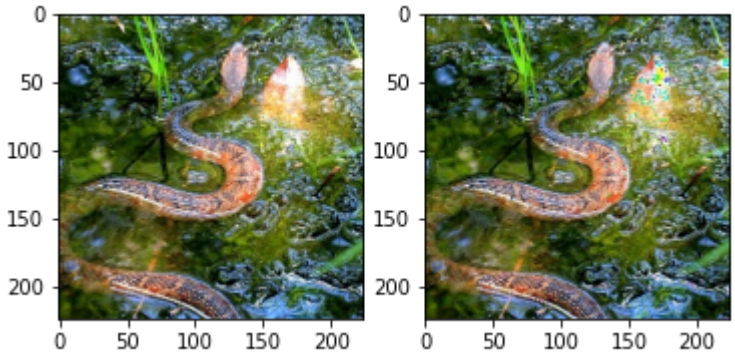
```
        fc_out = model(input_image)
        fc_out = F.softmax(fc_out, dim=1)

        optimizer.zero_grad()
        loss = criteron(fc_out, target_variable)

        loss.backward(retain_graph = True)
    #     grads = torch.sign(img_variable.grad.data)
    #     print(img_variable.grad.data)
    #     optimizer.zero_grad()
        optimizer.step()
    #     epsilon = [0.0001, 0.001, 0.0013, 0.009, 0.015, 0.120, 0.25, 0.302]
    #     for i in epsilon:
    #         img_variable = img_variable.data + i

        if iteration == 1 or iteration%50 == 0:
            print("Current Iteration: ",iteration,"\tCurrent Cost: ", loss)
    #         final_image = deprocess(img_variable)
    #         cv2.imwrite("temp.jpg", final_image)
    #         print(get_image_label("temp.jpg"))
    #         print(fc_out,target_variable)
    return input_image
```

In [79]:

```
targets=[5,10,15]
for url in urls:
    img = preprocess(url[1])
    img_tensor = torch.from_numpy(img)
    img_tensor = img_tensor.unsqueeze_(0)
    pre_image = deprocess(img_tensor)
    for target in targets:
        pre_processed="pre_processed"+url[1]+"-"+str(target)+".jpg"
        cv2.imwrite(pre_processed, pre_image)
        img = preprocess(url[1])
        img_tensor = torch.from_numpy(img)
        img_tensor = img_tensor.unsqueeze_(0)
        img_variable = Variable(img_tensor, requires_grad=True)
        target_vector = np.array([target], dtype=np.int64)
        target_tensor = torch.from_numpy(target_vector)
        target_variable = Variable(target_tensor)
#         adv_image=generate_adverserial(img_variable,target_variable)
#         adv_image=generate_adverserial(img_variable,url[2])
        adv_image=generate_adverserial(img_variable,target_variable,30)
        final_image = deprocess(adv_image)
        advout="adverseial_"+url[1]+"-"+str(target)+".jpg"
        cv2.imwrite(advout, final_image)
        print("original:",get_image_label(pre_processed))
        print("adverserial:",get_image_label(advout))
        f = plt.figure()
        f.add_subplot(1,2, 1)
        plt.imshow(cv2.cvtColor(cv2.imread(pre_processed),cv2.COLOR_BGR2RGB))
        f.add_subplot(1,2, 2)
        plt.imshow(cv2.cvtColor(cv2.imread(advout),cv2.COLOR_BGR2RGB))
        plt.show(block=True)
```

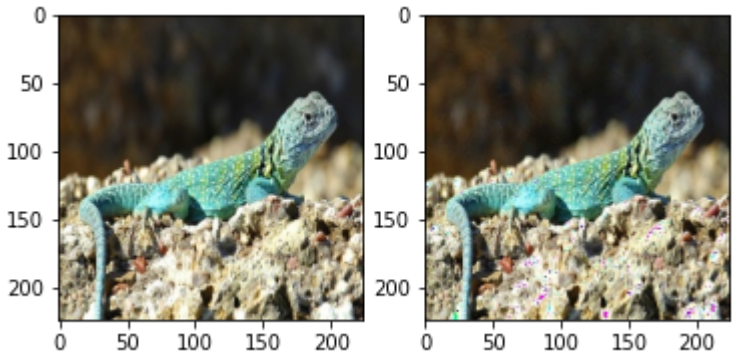
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9094, grad_fn=<NllLossBackward>)
original: (tensor(58), 'water snake')
adverserial: (tensor(30), 'bullfrog')
```

```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9094, grad_fn=<NllLossBackward>)
original: (tensor(58), 'water snake')
adverserial: (tensor(58), 'water snake')
```



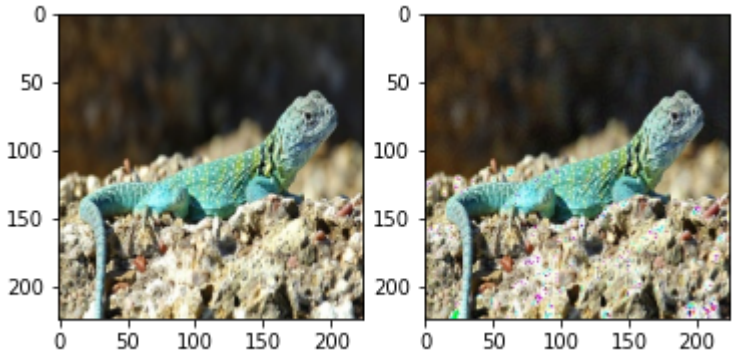
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9094, grad_fn=<NllLossBackward>)
original: (tensor(58), 'water snake')
adverserial: (tensor(15), 'robin')
```

```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9090, grad_fn=<NllLossBackward>)
original: (tensor(42), 'agama')
adverserial: (tensor(300), 'tiger beetle')
```

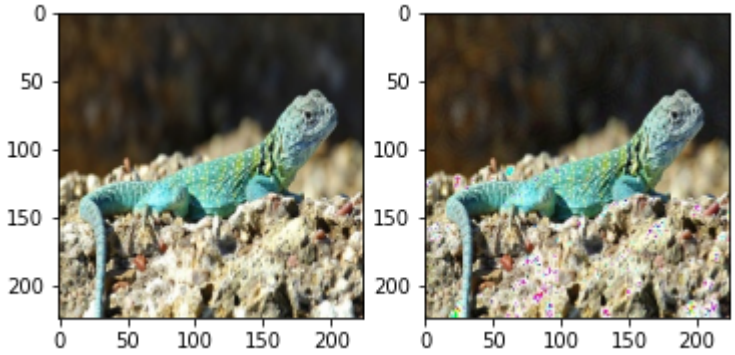


```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9090, grad_fn=<NllLossBackward>)
```

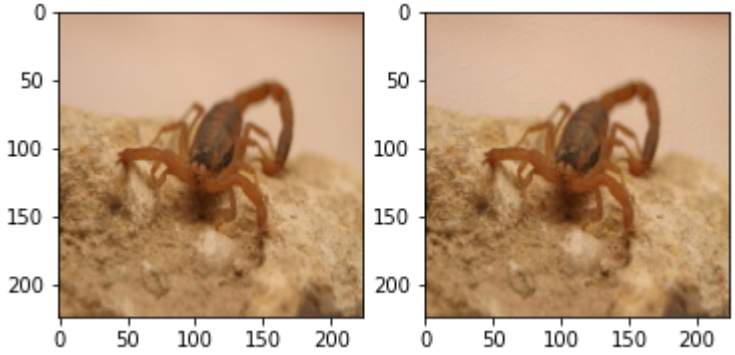
```
original: (tensor(42), 'agama')
adverserial: (tensor(10), 'brambling')
```



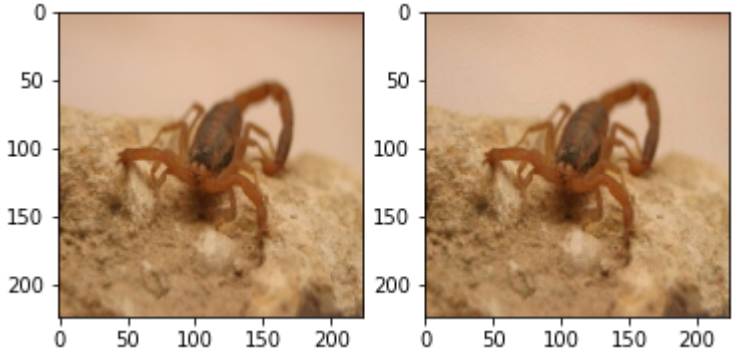
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9090, grad_fn=<NllLossBackward>)
original: (tensor(42), 'agama')
adverserial: (tensor(15), 'robin')
```



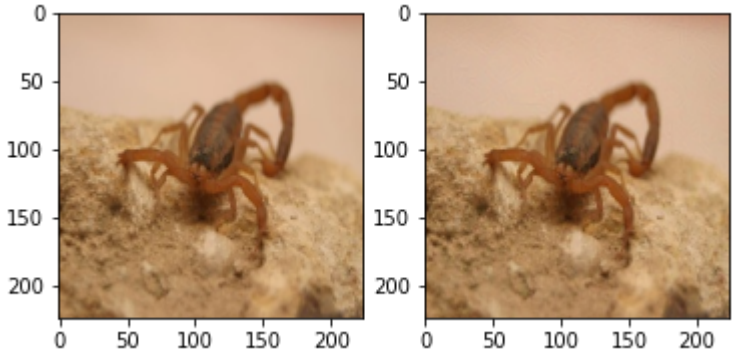
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(71), 'scorpion')
adverserial: (tensor(5), 'electric ray')
```



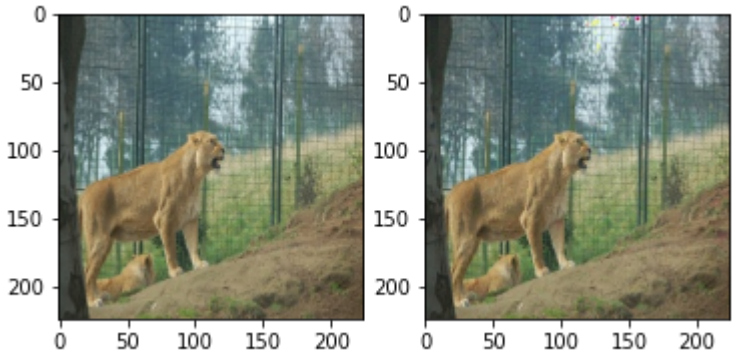
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(71), 'scorpion')
adverserial: (tensor(73), 'barn spider')
```



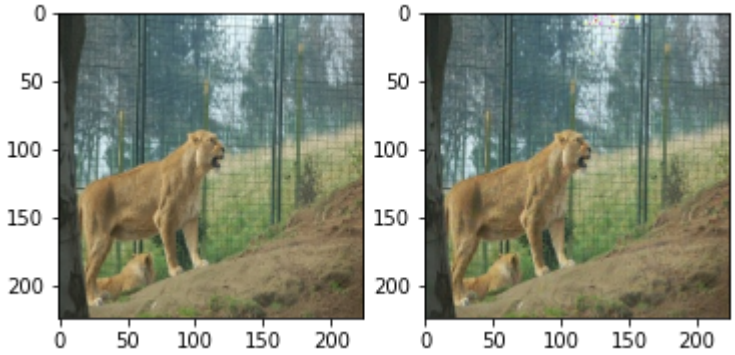
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(71), 'scorpion')
adverserial: (tensor(73), 'barn spider')
```



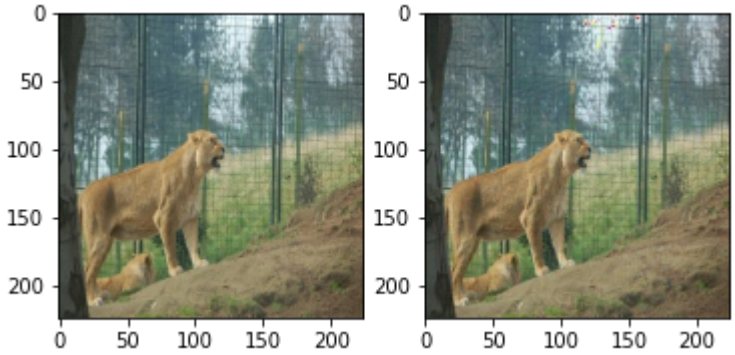
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9089, grad_fn=<NllLossBackward>)
original: (tensor(291), 'lion')
adverserial: (tensor(48), 'Komodo dragon')
```



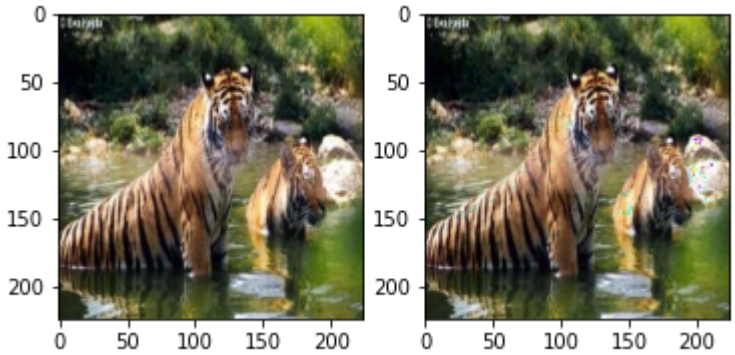
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9089, grad_fn=<NllLossBackward>)
original: (tensor(291), 'lion')
adverserial: (tensor(10), 'brambling')
```



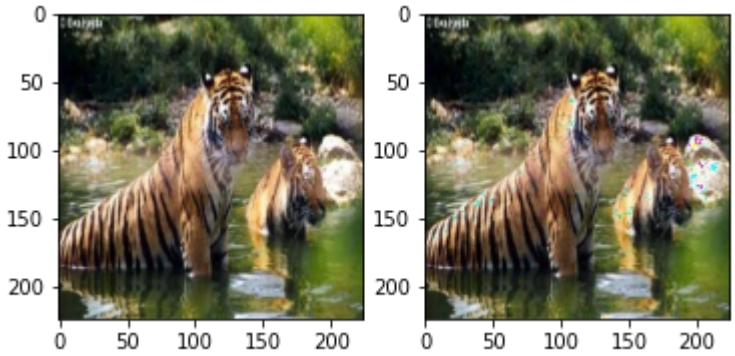
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9089, grad_fn=<NllLossBackward>)
original: (tensor(291), 'lion')
adverserial: (tensor(15), 'robin')
```



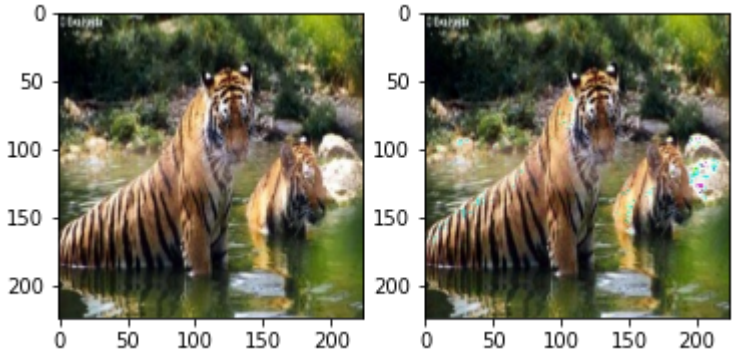
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(292), 'tiger')
adverserial: (tensor(340), 'zebra')
```



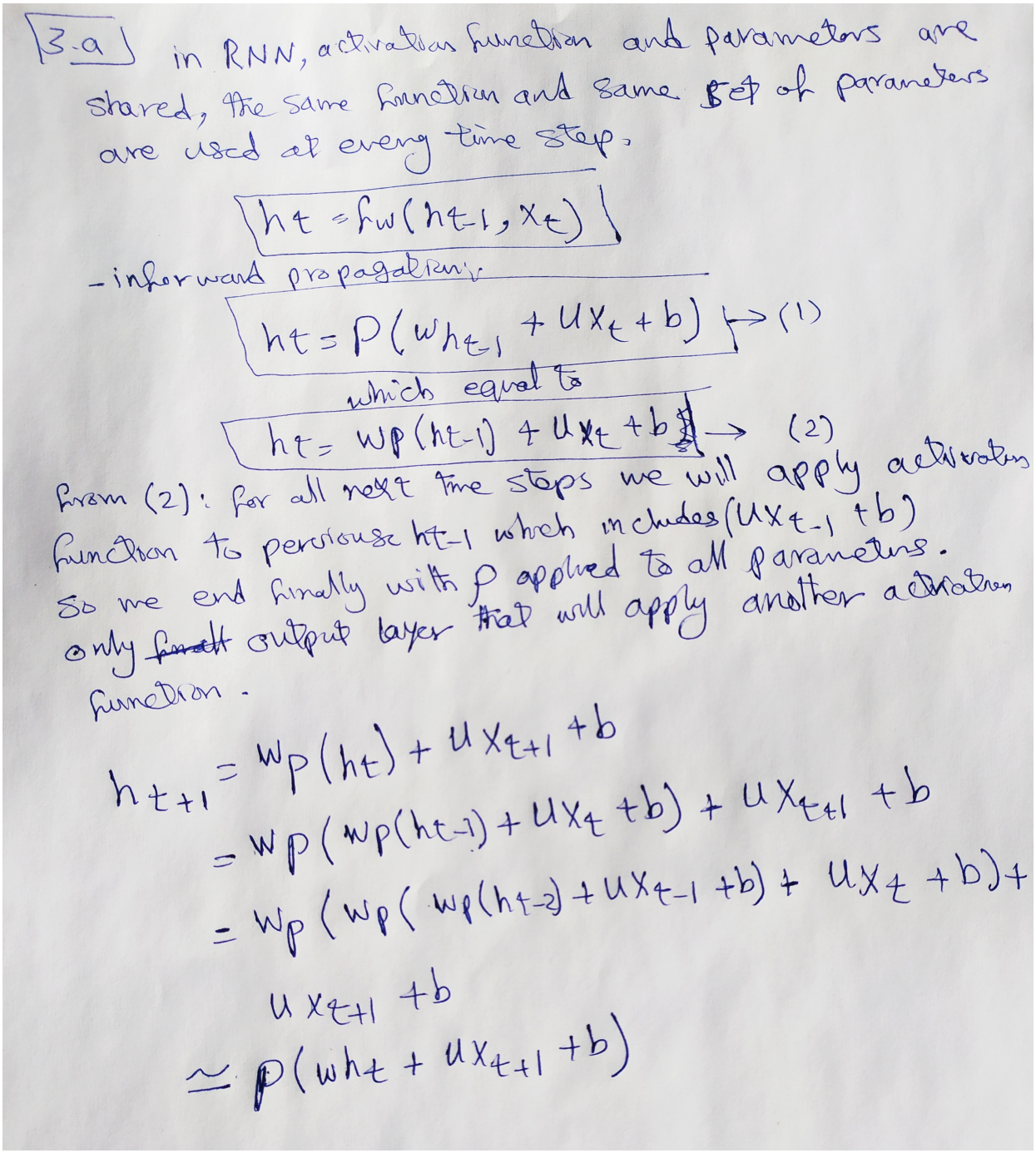
```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(292), 'tiger')
adverserial: (tensor(396), 'lionfish')
```



```
Starting Optimization...
Current Iteration:  1   Current Cost:  tensor(6.9093, grad_fn=<NllLossBackward>)
original: (tensor(292), 'tiger')
adverserial: (tensor(15), 'robin')
```



3.B



In [190…

```
import numpy as np
np.random.seed(0)
class RecurrentNetwork(object):
    """W_hh means a weight matrix that accepts a hidden state and produce a new hidden
    Similarly, W_xh represents a weight matrix that accepts an input vector and produce
    """
    def __init__(self):
        self.hidden_state = np.zeros((3, 3))
        self.hidden_state2 = np.zeros((3, 3))
        self.W_hh = np.random.randn(3, 3)
        self.W_xh = np.random.randn(3, 3)
        self.W_hy = np.random.randn(3, 3)
```

```
        self.Bh = np.random.randn(3,)
        self.By = np.random.rand(3,)

    def forward_prop(self, x):
        # The order of which you do dot product is entirely up to you. The gradient upd
        # as long as the matrix dimension matches up.
        self.hidden_state = np.tanh(np.dot(self.hidden_state, self.W_hh) + np.dot(x, se
#         print(self.W_hy.dot(self.hidden_state) + self.By)
        self.hidden_state2= np.tanh(np.dot(self.hidden_state2, self.W_hh)) + np.dot(x,
#         print(self.W_hy.dot(self.hidden_state2) + self.By)
        return self.W_hy.dot(self.hidden_state) + self.By,self.W_hy.dot(self.hidden_sta

input_vector = np.ones((3, 3))
RNN = RecurrentNetwork()

# Notice that same input, but leads to different ouptut at every single time step.
print(RNN.forward_prop(input_vector))
print(RNN.forward_prop(input_vector))
print(RNN.forward_prop(input_vector))
```

```
(array([[-1.73665315, -2.40366542, -2.72344361],
       [ 1.61591482,  1.45557046,  1.13262256],
       [ 1.68977504,  1.54059305,  1.21757531]]), array([[-3.13455948, -9.50547134, -9.4
2482   ],
       [ 1.96646911,  3.23649703,  2.81313303],
       [ 2.07884955,  3.51721453,  3.08274661]]))
(array([[-2.15023381, -2.41205828, -2.71701457],
       [ 1.71962883,  1.45767515,  1.13101034],
       [ 1.80488553,  1.542929  ,  1.21578594]]), array([[ -6.22857733, -12.5994636 ,  -
6.39824394],
       [  2.74235888,   4.01238038,   2.0541557 ],
       [  2.93999702,   4.37835488,   2.24036999]]))
(array([[-2.15024751, -2.41207375, -2.720968  ],
       [ 1.71963227,  1.45767903,  1.13200175],
       [ 1.80488935,  1.54293331,  1.21688628]]), array([[ -6.22857733, -12.59948899,  -
6.41337362],
       [  2.74235888,   4.01238675,   2.05794978],
       [  2.93999702,   4.37836194,   2.24458098]]))
```

3.C

https://mmuratarat.github.io/2019-02-07/bptt-of-rnn

In vanilla RNNs, vanishing/exploding gradient comes from the repeated application of the recurrent connections. which happen because of recursive derivative we need to compute

$$\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j} = \frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{k+1}}{\partial h_k}$$

if we want to backpropagate through t-k timesteps, this gradient will be:

$$\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j} = \prod_{j=k}^{t} diag(\phi_h'(W_{xh}^T \cdot X_{j+1} + W_{hh}^T \cdot h_j + b_h)W_{hh}$$

if the dominant eigenvalue of the matrix Whh is greater than 1, the gradient explodes. If it is less than 1, the gradient vanishes.

The gradient $\frac{\partial h_{t+1}}{\partial h_k}$ is a product of Jacobian matrices that are multiplied many times, $t-k$ times in our case:

$$\left\| \frac{\partial h_{t+1}}{\partial h_k} \right\| = \left\| \prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j} \right\| \leq (\gamma_W \gamma_h)^{t-k}$$

This can become very small or very large quickly, and the locality assumption of gradient descent breaks down as the sequence gets longer (i.e the distance between t and k increases). Then the value of γ will determine if the gradient either gets very large (explodes) or gets very small (vanishes).

Since γ is associated with the leading eigenvalues of $\frac{\partial h_{j+1}}{\partial h_j}$, the recursive product of $t-k$ Jacobian matrices makes it possible to influence the overall gradient in such a way that for $\gamma < 1$ the gradient tends to vanish while for $\gamma > 1$ the gradient tends to explode.

3.D

gradient tends to vanish when Y<1 and tends to explode when Y>1.

4.B

Self attention sees its input as a set, not a sequence. If we permute the input sequence, the output sequence will be exactly the same, except permuted also (i.e. self-attention is permutation equivariant). We will mitigate this somewhat when we build the full transformer, but the self-attention by itself actually ignores the sequential nature of the input.
http://peterbloem.nl/blog/transformers

In [79]:
```
from scipy.special import softmax
import itertools
import numpy as np
from copy import copy, deepcopy
T=5
D=5
p0=range(0,T)
permutations=list(itertools.permutations(p0))
X = np.random.randint(10,size=(T, D))
Wq=np.random.rand(D, D)
Wk=np.random.rand(D, D)
Wv=np.random.rand(D, D)
# print(X)
def s_x(x):
    Q=x@Wq
    # print(Wq)
    K=x@Wk
    # print(Wk)
    V=x@Wv
    # print(X)
    alpha=2
    A=(Q@K.transpose())/alpha
    s=softmax(A)@V
```

```
    return s

for p in permutations:
    psx=deepcopy(s_x(X))
    psx[p0]=psx[list(p)]
    px=deepcopy(X)
    px[p0]=px[list(p)]
    spx=s_x(px)
    print('p:',p,'-> psx==spx:',np.allclose(psx,spx))
```

```
p: (0, 1, 2, 3, 4) -> psx==spx: True
p: (0, 1, 2, 4, 3) -> psx==spx: True
p: (0, 1, 3, 2, 4) -> psx==spx: True
p: (0, 1, 3, 4, 2) -> psx==spx: True
p: (0, 1, 4, 2, 3) -> psx==spx: True
p: (0, 1, 4, 3, 2) -> psx==spx: True
p: (0, 2, 1, 3, 4) -> psx==spx: True
p: (0, 2, 1, 4, 3) -> psx==spx: True
p: (0, 2, 3, 1, 4) -> psx==spx: True
p: (0, 2, 3, 4, 1) -> psx==spx: True
p: (0, 2, 4, 1, 3) -> psx==spx: True
p: (0, 2, 4, 3, 1) -> psx==spx: True
p: (0, 3, 1, 2, 4) -> psx==spx: True
p: (0, 3, 1, 4, 2) -> psx==spx: True
p: (0, 3, 2, 1, 4) -> psx==spx: True
p: (0, 3, 2, 4, 1) -> psx==spx: True
p: (0, 3, 4, 1, 2) -> psx==spx: True
p: (0, 3, 4, 2, 1) -> psx==spx: True
p: (0, 4, 1, 2, 3) -> psx==spx: True
p: (0, 4, 1, 3, 2) -> psx==spx: True
p: (0, 4, 2, 1, 3) -> psx==spx: True
p: (0, 4, 2, 3, 1) -> psx==spx: True
p: (0, 4, 3, 1, 2) -> psx==spx: True
p: (0, 4, 3, 2, 1) -> psx==spx: True
p: (1, 0, 2, 3, 4) -> psx==spx: True
p: (1, 0, 2, 4, 3) -> psx==spx: True
p: (1, 0, 3, 2, 4) -> psx==spx: True
p: (1, 0, 3, 4, 2) -> psx==spx: True
p: (1, 0, 4, 2, 3) -> psx==spx: True
p: (1, 0, 4, 3, 2) -> psx==spx: True
p: (1, 2, 0, 3, 4) -> psx==spx: True
p: (1, 2, 0, 4, 3) -> psx==spx: True
p: (1, 2, 3, 0, 4) -> psx==spx: True
p: (1, 2, 3, 4, 0) -> psx==spx: True
p: (1, 2, 4, 0, 3) -> psx==spx: True
p: (1, 2, 4, 3, 0) -> psx==spx: True
p: (1, 3, 0, 2, 4) -> psx==spx: True
p: (1, 3, 0, 4, 2) -> psx==spx: True
p: (1, 3, 2, 0, 4) -> psx==spx: True
p: (1, 3, 2, 4, 0) -> psx==spx: True
p: (1, 3, 4, 0, 2) -> psx==spx: True
p: (1, 3, 4, 2, 0) -> psx==spx: True
p: (1, 4, 0, 2, 3) -> psx==spx: True
p: (1, 4, 0, 3, 2) -> psx==spx: True
p: (1, 4, 2, 0, 3) -> psx==spx: True
p: (1, 4, 2, 3, 0) -> psx==spx: True
p: (1, 4, 3, 0, 2) -> psx==spx: True
p: (1, 4, 3, 2, 0) -> psx==spx: True
p: (2, 0, 1, 3, 4) -> psx==spx: True
p: (2, 0, 1, 4, 3) -> psx==spx: True
p: (2, 0, 3, 1, 4) -> psx==spx: True
p: (2, 0, 3, 4, 1) -> psx==spx: True
p: (2, 0, 4, 1, 3) -> psx==spx: True
```

```
p: (2, 0, 4, 3, 1) -> psx==spx: True
p: (2, 1, 0, 3, 4) -> psx==spx: True
p: (2, 1, 0, 4, 3) -> psx==spx: True
p: (2, 1, 3, 0, 4) -> psx==spx: True
p: (2, 1, 3, 4, 0) -> psx==spx: True
p: (2, 1, 4, 0, 3) -> psx==spx: True
p: (2, 1, 4, 3, 0) -> psx==spx: True
p: (2, 3, 0, 1, 4) -> psx==spx: True
p: (2, 3, 0, 4, 1) -> psx==spx: True
p: (2, 3, 1, 0, 4) -> psx==spx: True
p: (2, 3, 1, 4, 0) -> psx==spx: True
p: (2, 3, 4, 0, 1) -> psx==spx: True
p: (2, 3, 4, 1, 0) -> psx==spx: True
p: (2, 4, 0, 1, 3) -> psx==spx: True
p: (2, 4, 0, 3, 1) -> psx==spx: True
p: (2, 4, 1, 0, 3) -> psx==spx: True
p: (2, 4, 1, 3, 0) -> psx==spx: True
p: (2, 4, 3, 0, 1) -> psx==spx: True
p: (2, 4, 3, 1, 0) -> psx==spx: True
p: (3, 0, 1, 2, 4) -> psx==spx: True
p: (3, 0, 1, 4, 2) -> psx==spx: True
p: (3, 0, 2, 1, 4) -> psx==spx: True
p: (3, 0, 2, 4, 1) -> psx==spx: True
p: (3, 0, 4, 1, 2) -> psx==spx: True
p: (3, 0, 4, 2, 1) -> psx==spx: True
p: (3, 1, 0, 2, 4) -> psx==spx: True
p: (3, 1, 0, 4, 2) -> psx==spx: True
p: (3, 1, 2, 0, 4) -> psx==spx: True
p: (3, 1, 2, 4, 0) -> psx==spx: True
p: (3, 1, 4, 0, 2) -> psx==spx: True
p: (3, 1, 4, 2, 0) -> psx==spx: True
p: (3, 2, 0, 1, 4) -> psx==spx: True
p: (3, 2, 0, 4, 1) -> psx==spx: True
p: (3, 2, 1, 0, 4) -> psx==spx: True
p: (3, 2, 1, 4, 0) -> psx==spx: True
p: (3, 2, 4, 0, 1) -> psx==spx: True
p: (3, 2, 4, 1, 0) -> psx==spx: True
p: (3, 4, 0, 1, 2) -> psx==spx: True
p: (3, 4, 0, 2, 1) -> psx==spx: True
p: (3, 4, 1, 0, 2) -> psx==spx: True
p: (3, 4, 1, 2, 0) -> psx==spx: True
p: (3, 4, 2, 0, 1) -> psx==spx: True
p: (3, 4, 2, 1, 0) -> psx==spx: True
p: (4, 0, 1, 2, 3) -> psx==spx: True
p: (4, 0, 1, 3, 2) -> psx==spx: True
p: (4, 0, 2, 1, 3) -> psx==spx: True
p: (4, 0, 2, 3, 1) -> psx==spx: True
p: (4, 0, 3, 1, 2) -> psx==spx: True
p: (4, 0, 3, 2, 1) -> psx==spx: True
p: (4, 1, 0, 2, 3) -> psx==spx: True
p: (4, 1, 0, 3, 2) -> psx==spx: True
p: (4, 1, 2, 0, 3) -> psx==spx: True
p: (4, 1, 2, 3, 0) -> psx==spx: True
p: (4, 1, 3, 0, 2) -> psx==spx: True
p: (4, 1, 3, 2, 0) -> psx==spx: True
p: (4, 2, 0, 1, 3) -> psx==spx: True
p: (4, 2, 0, 3, 1) -> psx==spx: True
p: (4, 2, 1, 0, 3) -> psx==spx: True
p: (4, 2, 1, 3, 0) -> psx==spx: True
p: (4, 2, 3, 0, 1) -> psx==spx: True
p: (4, 2, 3, 1, 0) -> psx==spx: True
p: (4, 3, 0, 1, 2) -> psx==spx: True
p: (4, 3, 0, 2, 1) -> psx==spx: True
p: (4, 3, 1, 0, 2) -> psx==spx: True
p: (4, 3, 1, 2, 0) -> psx==spx: True
```

```
p: (4, 3, 2, 0, 1) -> psx==spx: True
p: (4, 3, 2, 1, 0) -> psx==spx: True
```

4.C apply average function

In [194…

```
from skimage.measure import block_reduce
print(X)
for p in permutations:
# p= permutations[2]
    px=deepcopy(X)
    px[p0]=px[list(p)]
    spx=s_x(px)
    sx=s_x(X)

    sx_mean=np.average(sx)
#     print(sx_mean)
#     spx_mean=block_reduce(spx, (3,3), np.mean)
    spx_mean=np.average(spx)
#     print(spx_mean)
    print('p:',p,'-> sx_avg==spx_avg:',np.allclose(sx_mean,spx_mean))
```

```
[[6 7 7 0 0]
 [1 4 5 1 0]
 [8 1 0 0 1]
 [6 6 9 5 6]
 [6 3 2 2 3]]
p: (0, 1, 2, 3, 4) -> sx_avg==spx_avg: True
p: (0, 1, 2, 4, 3) -> sx_avg==spx_avg: True
p: (0, 1, 3, 2, 4) -> sx_avg==spx_avg: True
p: (0, 1, 3, 4, 2) -> sx_avg==spx_avg: True
p: (0, 1, 4, 2, 3) -> sx_avg==spx_avg: True
p: (0, 1, 4, 3, 2) -> sx_avg==spx_avg: True
p: (0, 2, 1, 3, 4) -> sx_avg==spx_avg: True
p: (0, 2, 1, 4, 3) -> sx_avg==spx_avg: True
p: (0, 2, 3, 1, 4) -> sx_avg==spx_avg: True
p: (0, 2, 3, 4, 1) -> sx_avg==spx_avg: True
p: (0, 2, 4, 1, 3) -> sx_avg==spx_avg: True
p: (0, 2, 4, 3, 1) -> sx_avg==spx_avg: True
p: (0, 3, 1, 2, 4) -> sx_avg==spx_avg: True
p: (0, 3, 1, 4, 2) -> sx_avg==spx_avg: True
p: (0, 3, 2, 1, 4) -> sx_avg==spx_avg: True
p: (0, 3, 2, 4, 1) -> sx_avg==spx_avg: True
p: (0, 3, 4, 1, 2) -> sx_avg==spx_avg: True
p: (0, 3, 4, 2, 1) -> sx_avg==spx_avg: True
p: (0, 4, 1, 2, 3) -> sx_avg==spx_avg: True
p: (0, 4, 1, 3, 2) -> sx_avg==spx_avg: True
p: (0, 4, 2, 1, 3) -> sx_avg==spx_avg: True
p: (0, 4, 2, 3, 1) -> sx_avg==spx_avg: True
p: (0, 4, 3, 1, 2) -> sx_avg==spx_avg: True
p: (0, 4, 3, 2, 1) -> sx_avg==spx_avg: True
p: (1, 0, 2, 3, 4) -> sx_avg==spx_avg: True
p: (1, 0, 2, 4, 3) -> sx_avg==spx_avg: True
p: (1, 0, 3, 2, 4) -> sx_avg==spx_avg: True
p: (1, 0, 3, 4, 2) -> sx_avg==spx_avg: True
p: (1, 0, 4, 2, 3) -> sx_avg==spx_avg: True
p: (1, 0, 4, 3, 2) -> sx_avg==spx_avg: True
p: (1, 2, 0, 3, 4) -> sx_avg==spx_avg: True
p: (1, 2, 0, 4, 3) -> sx_avg==spx_avg: True
p: (1, 2, 3, 0, 4) -> sx_avg==spx_avg: True
p: (1, 2, 3, 4, 0) -> sx_avg==spx_avg: True
p: (1, 2, 4, 0, 3) -> sx_avg==spx_avg: True
p: (1, 2, 4, 3, 0) -> sx_avg==spx_avg: True
```

```
p: (1, 3, 0, 2, 4) -> sx_avg==spx_avg: True
p: (1, 3, 0, 4, 2) -> sx_avg==spx_avg: True
p: (1, 3, 2, 0, 4) -> sx_avg==spx_avg: True
p: (1, 3, 2, 4, 0) -> sx_avg==spx_avg: True
p: (1, 3, 4, 0, 2) -> sx_avg==spx_avg: True
p: (1, 3, 4, 2, 0) -> sx_avg==spx_avg: True
p: (1, 4, 0, 2, 3) -> sx_avg==spx_avg: True
p: (1, 4, 0, 3, 2) -> sx_avg==spx_avg: True
p: (1, 4, 2, 0, 3) -> sx_avg==spx_avg: True
p: (1, 4, 2, 3, 0) -> sx_avg==spx_avg: True
p: (1, 4, 3, 0, 2) -> sx_avg==spx_avg: True
p: (1, 4, 3, 2, 0) -> sx_avg==spx_avg: True
p: (2, 0, 1, 3, 4) -> sx_avg==spx_avg: True
p: (2, 0, 1, 4, 3) -> sx_avg==spx_avg: True
p: (2, 0, 3, 1, 4) -> sx_avg==spx_avg: True
p: (2, 0, 3, 4, 1) -> sx_avg==spx_avg: True
p: (2, 0, 4, 1, 3) -> sx_avg==spx_avg: True
p: (2, 0, 4, 3, 1) -> sx_avg==spx_avg: True
p: (2, 1, 0, 3, 4) -> sx_avg==spx_avg: True
p: (2, 1, 0, 4, 3) -> sx_avg==spx_avg: True
p: (2, 1, 3, 0, 4) -> sx_avg==spx_avg: True
p: (2, 1, 3, 4, 0) -> sx_avg==spx_avg: True
p: (2, 1, 4, 0, 3) -> sx_avg==spx_avg: True
p: (2, 1, 4, 3, 0) -> sx_avg==spx_avg: True
p: (2, 3, 0, 1, 4) -> sx_avg==spx_avg: True
p: (2, 3, 0, 4, 1) -> sx_avg==spx_avg: True
p: (2, 3, 1, 0, 4) -> sx_avg==spx_avg: True
p: (2, 3, 1, 4, 0) -> sx_avg==spx_avg: True
p: (2, 3, 4, 0, 1) -> sx_avg==spx_avg: True
p: (2, 3, 4, 1, 0) -> sx_avg==spx_avg: True
p: (2, 4, 0, 1, 3) -> sx_avg==spx_avg: True
p: (2, 4, 0, 3, 1) -> sx_avg==spx_avg: True
p: (2, 4, 1, 0, 3) -> sx_avg==spx_avg: True
p: (2, 4, 1, 3, 0) -> sx_avg==spx_avg: True
p: (2, 4, 3, 0, 1) -> sx_avg==spx_avg: True
p: (2, 4, 3, 1, 0) -> sx_avg==spx_avg: True
p: (3, 0, 1, 2, 4) -> sx_avg==spx_avg: True
p: (3, 0, 1, 4, 2) -> sx_avg==spx_avg: True
p: (3, 0, 2, 1, 4) -> sx_avg==spx_avg: True
p: (3, 0, 2, 4, 1) -> sx_avg==spx_avg: True
p: (3, 0, 4, 1, 2) -> sx_avg==spx_avg: True
p: (3, 0, 4, 2, 1) -> sx_avg==spx_avg: True
p: (3, 1, 0, 2, 4) -> sx_avg==spx_avg: True
p: (3, 1, 0, 4, 2) -> sx_avg==spx_avg: True
p: (3, 1, 2, 0, 4) -> sx_avg==spx_avg: True
p: (3, 1, 2, 4, 0) -> sx_avg==spx_avg: True
p: (3, 1, 4, 0, 2) -> sx_avg==spx_avg: True
p: (3, 1, 4, 2, 0) -> sx_avg==spx_avg: True
p: (3, 2, 0, 1, 4) -> sx_avg==spx_avg: True
p: (3, 2, 0, 4, 1) -> sx_avg==spx_avg: True
p: (3, 2, 1, 0, 4) -> sx_avg==spx_avg: True
p: (3, 2, 1, 4, 0) -> sx_avg==spx_avg: True
p: (3, 2, 4, 0, 1) -> sx_avg==spx_avg: True
p: (3, 2, 4, 1, 0) -> sx_avg==spx_avg: True
p: (3, 4, 0, 1, 2) -> sx_avg==spx_avg: True
p: (3, 4, 0, 2, 1) -> sx_avg==spx_avg: True
p: (3, 4, 1, 0, 2) -> sx_avg==spx_avg: True
p: (3, 4, 1, 2, 0) -> sx_avg==spx_avg: True
p: (3, 4, 2, 0, 1) -> sx_avg==spx_avg: True
p: (3, 4, 2, 1, 0) -> sx_avg==spx_avg: True
p: (4, 0, 1, 2, 3) -> sx_avg==spx_avg: True
p: (4, 0, 1, 3, 2) -> sx_avg==spx_avg: True
p: (4, 0, 2, 1, 3) -> sx_avg==spx_avg: True
p: (4, 0, 2, 3, 1) -> sx_avg==spx_avg: True
p: (4, 0, 3, 1, 2) -> sx_avg==spx_avg: True
```

```
p: (4, 0, 3, 2, 1) -> sx_avg==spx_avg: True
p: (4, 1, 0, 2, 3) -> sx_avg==spx_avg: True
p: (4, 1, 0, 3, 2) -> sx_avg==spx_avg: True
p: (4, 1, 2, 0, 3) -> sx_avg==spx_avg: True
p: (4, 1, 2, 3, 0) -> sx_avg==spx_avg: True
p: (4, 1, 3, 0, 2) -> sx_avg==spx_avg: True
p: (4, 1, 3, 2, 0) -> sx_avg==spx_avg: True
p: (4, 2, 0, 1, 3) -> sx_avg==spx_avg: True
p: (4, 2, 0, 3, 1) -> sx_avg==spx_avg: True
p: (4, 2, 1, 0, 3) -> sx_avg==spx_avg: True
p: (4, 2, 1, 3, 0) -> sx_avg==spx_avg: True
p: (4, 2, 3, 0, 1) -> sx_avg==spx_avg: True
p: (4, 2, 3, 1, 0) -> sx_avg==spx_avg: True
p: (4, 3, 0, 1, 2) -> sx_avg==spx_avg: True
p: (4, 3, 0, 2, 1) -> sx_avg==spx_avg: True
p: (4, 3, 1, 0, 2) -> sx_avg==spx_avg: True
p: (4, 3, 1, 2, 0) -> sx_avg==spx_avg: True
p: (4, 3, 2, 0, 1) -> sx_avg==spx_avg: True
p: (4, 3, 2, 1, 0) -> sx_avg==spx_avg: True
```

4.D

The Positionwise Feedforward network brings in some non-linear to MultiHead Attension

In [171…

```
class PositionwiseFeedForward(nn.Module):
    """
    Does a Linear + RELU + Linear on each of the timesteps
    """
    def __init__(self,B,T,D):
        """
        Parameters:
            input_depth: Size of last dimension of input
            filter_size:Hidden size of the middle layer
            output_depth: Size last dimension of the final output
        """
        input_depth, filter_size, output_depth=B,T,D
        super(PositionwiseFeedForward, self).__init__()


        self.layers = nn.ModuleList([
                      torch.nn.Linear(input_depth, filter_size),
                      torch.nn.Linear(filter_size, filter_size),
                      torch.nn.Linear(filter_size, output_depth)
                    ])
        self.relu = nn.ReLU()
    def forward(self, inputs):
        x = inputs
        for i, layer in enumerate(self.layers):
            x = layer(x)
            if i < len(self.layers):
                x = self.relu(x)
        return x
```

In [172…

```
class PositionwiseFeedForward_Conv(nn.Module):
    """
    Does a Linear + RELU + Linear on each of the timesteps
    """
    def __init__(self,B,T,D):
        """
```

```
        Parameters:
            input_depth: Size of last dimension of input
            filter_size:Hidden size of the middle layer
            output_depth: Size last dimension of the final output
        """
        input_depth, filter_size, output_depth=B,T,D
        super(PositionwiseFeedForward_Conv, self).__init__()


        self.layers = nn.ModuleList([
                      torch.nn.Conv1d(input_depth, filter_size,1),
                      torch.nn.Conv1d(filter_size, filter_size,1),
                      torch.nn.Conv1d(filter_size, output_depth,1)
                    ])
        self.relu = nn.ReLU()
    def forward(self, inputs):
        x = inputs
        for i, layer in enumerate(self.layers):
            x = layer(x)
            if i < len(self.layers):
                x = self.relu(x)
        return x
```

In [177…

```
Z=torch.randn([4,3,4])
permuted_tensor = Z.permute(0,2,1).clone().contiguous()
pwff_conv=PositionwiseFeedForward_Conv(Z.shape[0],Z.shape[1],Z.shape[2])
tensor_pwff_conv=pwff_conv.forward(permuted_tensor)

pwff=PositionwiseFeedForward(Z.shape[0],Z.shape[1],Z.shape[2])
for i in range(len(pwff_conv.layers)):
    pwff.layers[i].weight= torch.nn.Parameter(pwff_conv.layers[i].weight.squeeze(2))
    pwff.layers[i].bias = torch.nn.Parameter(pwff_conv.layers[i].bias)
tensor_pwff=pwff.forward(Z).permute(0,2,1).clone().contiguous()

print('pwff Linear out=',tensor_pwff.detach())
print('pwff_Conv1D Out=',tensor_pwff_conv.detach())
print( 'pwff_Linear==spwff_Conv1D:',np.allclose(tensor_pwff.detach(),tensor_pwff_conv.d
```

```
pwff Linear out= tensor([[[0.1803, 0.2522, 0.1838],
         [0.0000, 0.0421, 0.0439],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0612, 0.0000]],

        [[0.1809, 0.1785, 0.2299],
         [0.0054, 0.0037, 0.0000],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0312]],

        [[0.2116, 0.2455, 0.1926],
         [0.0278, 0.0335, 0.1105],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0467, 0.0000]],

        [[0.1785, 0.2554, 0.2053],
         [0.0037, 0.0598, 0.0000],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0598, 0.0000]]])
pwff_Conv1D Out= tensor([[[0.1803, 0.2522, 0.1838],
         [0.0000, 0.0421, 0.0439],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0612, 0.0000]],
```

```
        [[0.1809, 0.1785, 0.2299],
         [0.0054, 0.0037, 0.0000],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0312]],

        [[0.2116, 0.2455, 0.1926],
         [0.0278, 0.0335, 0.1105],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0467, 0.0000]],

        [[0.1785, 0.2554, 0.2053],
         [0.0037, 0.0598, 0.0000],
         [0.0000, 0.0000, 0.0000],
         [0.0000, 0.0598, 0.0000]]])
pwff_Linear==spwff_Conv1D: True
```

In [ ]: