In [12]:
```
%matplotlib inline
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import torch.nn as nn
import torch
import requests
from io import BytesIO
import tqdm

bucky_urls = [
    "https://cdn.vox-cdn.com/thumbor/3yCbVJHk2QeCNc47W6yaR2ok-OE=/0x0:4298x2
    "https://www.ncaa.com/_flysystem/public-s3/styles/large_16x9/public-s3/i
    "https://chancellor.wisc.edu/content/uploads/2018/08/Bucky_Parade_unveil
    "https://upload.wikimedia.org/wikipedia/commons/6/68/BuckinghamUBadger.j
]
def get_image_from_url(url):
    response = requests.get(url,headers={'User-Agent': 'Mozilla/5.0'})
    img = Image.open(BytesIO(response.content))
    img = img.resize((128,128))
    return img

def prepare_image_for_torch(img):
    img_arr = np.asarray(img) / 255.
    img_arr = torch.from_numpy(img_arr).float()
    return img_arr
```

In [2]:
```
class AutoEncoder(nn.Module):
    def __init__(self, in_chans, hidden_chans=10):
        super().__init__()
        self.encoder = nn.Linear(in_chans, hidden_chans, bias=False)
        self.decoder = nn.Linear(hidden_chans, in_chans, bias=False)

        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=0.001)

    def forward(self, x):
        B,H,W,D = x.shape
        x = x.flatten(1)
        z = self.encoder(x)
        xpred = self.decoder(z)
        return xpred.view(B,H,W,D)
```

In [3]:
```
def loss(pred, y):
    """MSE loss"""
    return (pred - y).pow(2).mean()

def fit(model, loss, images, iterations=300):
    """
    Helper function to train a model.
```

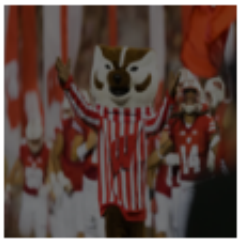
```
    - images: NxWxHx3 tensor
    - iterations: number of training iterations
    """
    opt = torch.optim.Adam(model.parameters(), lr=0.0005)
    for it in tqdm.tqdm(range(iterations)):
        pred = model(images)
        l = loss(pred, images)

        l.backward()
        opt.step()
        opt.zero_grad()
    return model

@torch.no_grad()
def display_reconstructions(model):
    """
    Helper function to visualize reconstructions obtained through the autoer
    """
    for i, url in enumerate(bucky_urls):
        img = get_image_from_url(url)
        img_arr = prepare_image_for_torch(img)
        img_rec = model(img_arr[None])[0]

        plt.subplot(len(bucky_urls),2,1+i*2)
        plt.imshow(img)
        plt.axis(False)
        plt.subplot(len(bucky_urls),2,2+i*2)
        plt.imshow(img_rec.numpy().clip(0, 1))
        plt.axis(False)
```

Question 2-a

In [4]:
```
# Declare the model
autoencoder = AutoEncoder(128*128*3, 10)

# Prepare training data
img = get_image_from_url(bucky_urls[0])
img_arr = prepare_image_for_torch(img)[None]

# Fit the model
fit(autoencoder, loss, img_arr)

# Display results
display_reconstructions(autoencoder)
```

```
100%|██████████| 300/300 [00:00<00:00, 304.24it/s]
```



Findings: It seems to have transformed each image into the first one and changes the brightness depending on the source image

In [13]:
```
# Part b)
# Step 1: Prepare all 3 images and stack them into a 3xHxWxD tensor
# Prepare training data
imgs = torch.stack([prepare_image_for_torch(get_image_from_url(url)) for url

# Step 2: Declare model, fit to data, visualize results
autoencoder = AutoEncoder(128*128*3, 10)
fit(autoencoder, loss, imgs)
display_reconstructions(autoencoder)
new_bucky_link = "https://en.wikipedia.org/wiki/File:BuckinghamUBadger.jpg"
```

```
100%|██████████| 300/300 [00:01<00:00, 217.16it/s]
```








Findings: All of the images are the same as the source image execpt the new image with appears to be a combination of all other images

(c) Why is the model not generalizing well to new images? Suggest two ideas to improve the generalization of the model.

Answer

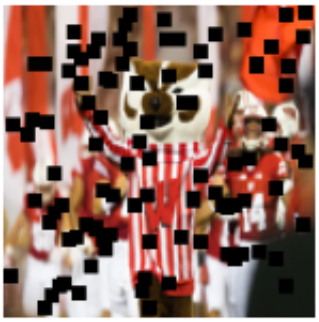
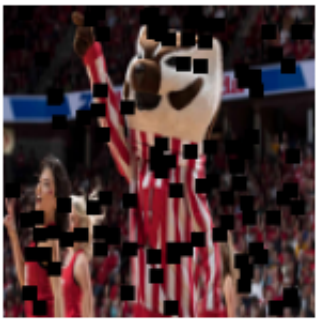
We dont have enough data to make the model generalize well to new images.

Ways to improve:

1. Increase the Dataset Size & Diversity
2. Increase models hidden channels/hidden layers

In [20]:
```
# Part d) and e)
# Step 1: Prepare all 3 images and stack them into a 3xHxWxD tensor
# Step 2: Rewrite fit_with_noise() so that, at each iteration, images are fi
# Step 3: Declare the model, fit to data, visualize results

def add_noise(img):
    img = img.clone()
    rows = img.shape[0]
    cols = img.shape[1]
    s = int(min(rows,cols)/20) # size of spot is 1/20 of smallest dimension

    for i in range(100):
        x = np.random.randint(cols-s)
        y = np.random.randint(rows-s)
```

```
        img[y:(y+s),x:(x+s)] = 0

    return img

for i, url in enumerate(bucky_urls[0:3]):
    img = get_image_from_url(url)
    img = prepare_image_for_torch(img)

    plt.subplot(1, 3, 1+i)
    plt.imshow(add_noise(img))
    plt.axis(False)
```


In [23]:

```
@torch.no_grad()
def display_reconstructions_with_noise(model):
    for i, url in enumerate(bucky_urls):
        img = get_image_from_url(url)
        img_arr = prepare_image_for_torch(img)
        img_noise = add_noise(img_arr)
        img_rec = model(img_noise[None])[0]

        plt.subplot(len(bucky_urls),3,1+i*3)
        plt.imshow(img)
        plt.axis(False)
        plt.subplot(len(bucky_urls),3,2+i*3)
        plt.imshow(img_noise.numpy().clip(0, 1))
        plt.axis(False)
        plt.subplot(len(bucky_urls),3,3+i*3)
        plt.imshow(img_rec.numpy().clip(0, 1))
        plt.axis(False)

def fit_with_noise(model, loss_fn, images, optimizer, epochs=300):
    model.train()
    for epoch in range(epochs):
        noisy_imgs = torch.stack([add_noise(img) for img in images])  # Appl
        optimizer.zero_grad()
        recon_imgs = model(noisy_imgs)  # Forward pass
        loss = loss_fn(recon_imgs, images)  # Compare with clean images
        loss.backward()
        optimizer.step()

        if epoch % 50 == 0:
            print(f"Epoch {epoch}/{epochs}, Loss: {loss.item():.6f}")
```

```
autoencoder = AutoEncoder(128 * 128 * 3, 10)  # Ensure correct model archite
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=0.001)
loss_fn = torch.nn.MSELoss()

# Reshape images into a flattened batch (if needed)
imgs = torch.stack([prepare_image_for_torch(get_image_from_url(url)) for url

fit_with_noise(autoencoder, loss_fn, imgs, optimizer, epochs=300)

display_reconstructions_with_noise(autoencoder)
```

```
Epoch 0/300, Loss: 0.266351
Epoch 50/300, Loss: 0.001055
Epoch 100/300, Loss: 0.001988
Epoch 150/300, Loss: 0.001066
Epoch 200/300, Loss: 0.000649
Epoch 250/300, Loss: 0.000541
```

In [ ]: