neural-networks-visualization

November 5, 2020

1 Neural Networks Visualization

In [1]:
```
import os
import random

import torch
from torch.nn import functional as F
import torchvision
import torchvision.transforms as T
import numpy as np
from scipy.ndimage.filters import gaussian_filter1d
import matplotlib.pyplot as plt
from PIL import Image

torchvision.models.vgg.model_urls[
    "squeezenet1_1"] = "http://webia.lip6.fr/~douillard/rdfia/squeezenet1_1-f364aa15.pt
os.environ["TORCH_HOME"] = "/tmp/torch"

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'viridis'
```

1.1 Fonctions et variables utiles

In [2]:
```
SQUEEZENET_MEAN = np.array([0.485, 0.456, 0.406], dtype=np.float32)
SQUEEZENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)

def preprocess(img, size=224):
    transform = T.Compose([
        T.Resize((size, size)),
        T.ToTensor(),
        T.Normalize(mean=SQUEEZENET_MEAN.tolist(),
                    std=SQUEEZENET_STD.tolist()),
        T.Lambda(lambda x: x[None]),
    ])
    return transform(img)
```

```
def deprocess(img, should_rescale=True):
    transform = T.Compose([
        T.Lambda(lambda x: x[0]),
        T.Normalize(mean=[0, 0, 0], std=(1.0 / SQUEEZENET_STD).tolist()),
        T.Normalize(mean=(-SQUEEZENET_MEAN).tolist(), std=[1, 1, 1]),
        T.Lambda(rescale) if should_rescale else T.Lambda(lambda x: x),
        T.ToPILImage(),
    ])
    return transform(img)

def rescale(x):
    low, high = x.min(), x.max()
    x_rescaled = (x - low) / (high - low)
    return x_rescaled

def blur_image(X_np, sigma=1):
    X_np = gaussian_filter1d(X_np, sigma, axis=2)
    X_np = gaussian_filter1d(X_np, sigma, axis=3)
    return torch.tensor(X_np)

def jitter(X, ox, oy):
    """
    Helper function to randomly jitter an image.

    Inputs
    - X: PyTorch Tensor of shape (N, C, H, W)
    - ox, oy: Integers giving number of pixels to jitter along W and H axes

    Returns: A new PyTorch Tensor of shape (N, C, H, W)
    """
    if ox != 0:
        left = X[:, :, :, :-ox]
        right = X[:, :, :, -ox:]
        X = torch.cat([right, left], dim=3)
    if oy != 0:
        top = X[:, :, :-oy]
        bottom = X[:, :, -oy:]
        X = torch.cat([bottom, top], dim=2)
    return X
```

1.2 Chargement du modèle

On utilisera le modèle Squeezenet qui est une modèle léger pré-appris sur ImageNet. Ce modèle sera figé puisque le but n'est pas de modifier/apprendre ses poids mais de les étudier.

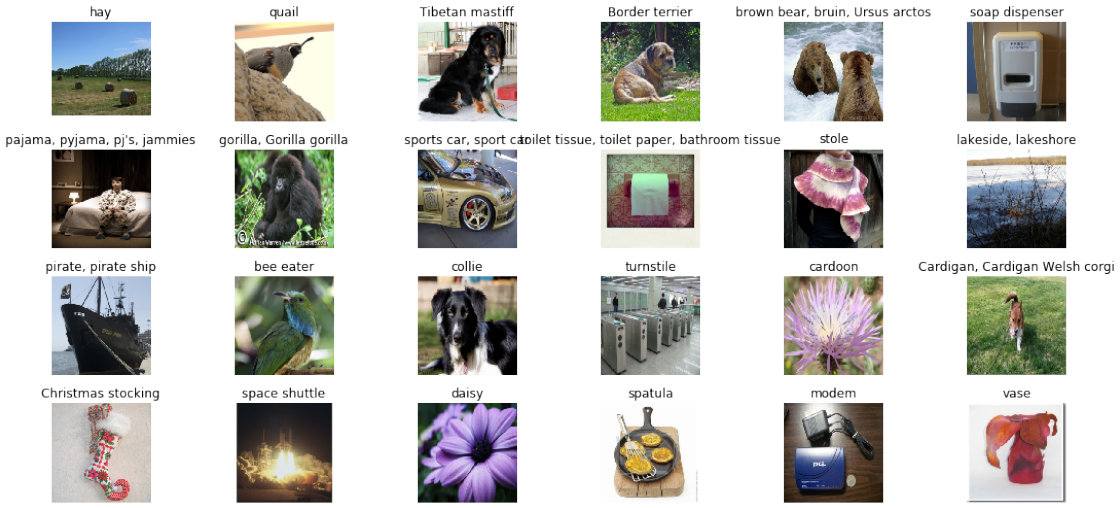
```
In [3]: # Chargement du modèle
        model = torchvision.models.squeezenet1_1(pretrained=True)
```

```
# Modele en mode test
model.eval()

# Freeze les poids
for param in model.parameters():
    param.requires_grad = False
```

1.3 Chargement d'images d'exemples

Permet de chargement dans les variables `X, y, class_names` 25 exemples de l'ensemble de validation d'ImageNet. `X` contient les images, `y` l'indice de la classe de chaque images, et `class_names` un dictionnaire permet d'obtenir le nom d'une classe à partir de son numéro.

```
In [4]: f = np.load("imagenet_val_25.npz", allow_pickle=True)
        X, y, class_names = f["X"], f["y"], f["label_map"].item()

        plt.figure(figsize=(15, 7))
        for i in range(24):
            plt.subplot(4, 6, i + 1)
            plt.imshow(X[i])
            plt.title(class_names[y[i]])
            plt.axis('off')
        plt.gcf().tight_layout()
```




```
In [5]: type(X), type(y), type(class_names)

Out[5]: (numpy.ndarray, numpy.ndarray, dict)

In [6]: len(X), len(y), len(class_names)

Out[6]: (25, 25, 999)
```

2 Saliency Maps

Conseil : Pour sélectionner 1 valeur particulière pour chaque ligne d'une matrice, vous pouvez faire comme cela :

```
x = torch.Tensor([[0.1, 0.0, 0.5, 0.1, 0.1],
                  [0.0, 0.1, 0.0, 0.6, 0.2],
                  [0.7, 0.1, 0.1, 0.3, 0.0]])
x[np.arange(3), [2, 3, 0]]
# 0.5000
# 0.6000
# 0.7000
#[torch.FloatTensor of size 3]
```

```
In [7]: def compute_saliency_maps(X, y, model):
            """
            Compute a class saliency map using the model for images X and labels y.

            Input:
            - X: Input images; Tensor of shape (N, 3, H, W)
            - y: Labels for X; LongTensor of shape (N,)
            - model: A pretrained CNN that will be used to compute the saliency map.

            Returns:
            - saliency: A Tensor of shape (N, H, W) giving the saliency maps for the input
            images.
            """
            X.requires_grad=True

            ##############################################################################
            # TODO: Implement this function. Perform a forward and backward pass through #
            # the model to compute the gradient of the correct class score with respect  #
            # to each input image. You first want to compute the loss over the correct   #
            # scores, and then compute the gradients with a backward pass.               #
            ##############################################################################

            yo_batch = model.forward(X)

            yo_batch = yo_batch[np.arange(len(y)), y]

            (yo_batch.sum()).backward()

            saliency = torch.abs(X.grad.data)
            saliency = torch.max(saliency, dim=1)[0]

            ##############################################################################
            #                               END OF YOUR CODE                             #
            ##############################################################################
            return saliency
```

Testez votre code avec la fonction ci-dessous :

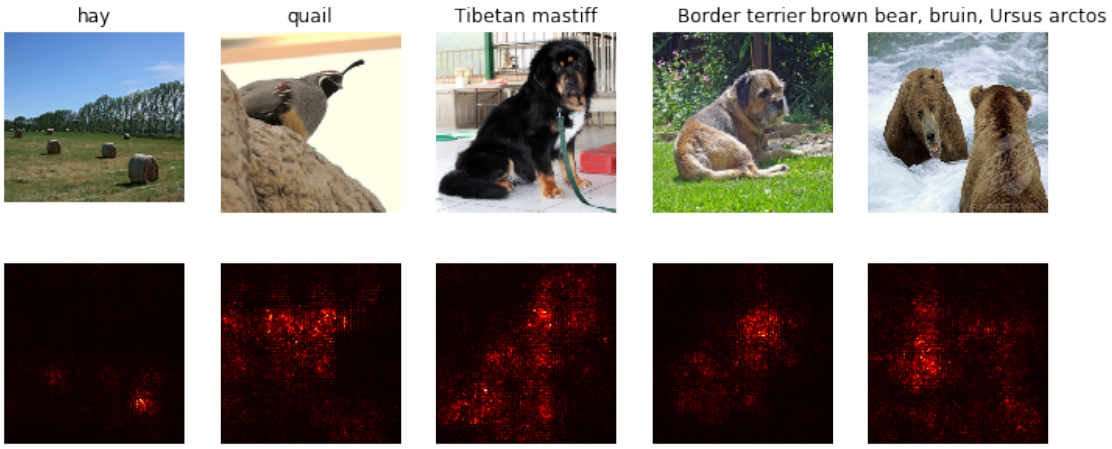
```
In [8]: model
```

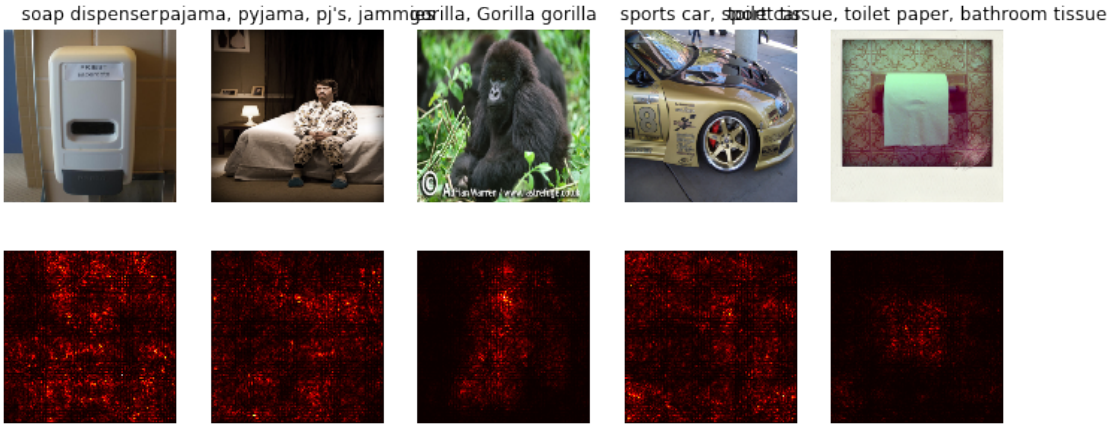
```
Out[8]: SqueezeNet(
          (features): Sequential(
            (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2))
            (1): ReLU(inplace=True)
            (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
            (3): Fire(
              (squeeze): Conv2d(64, 16, kernel_size=(1, 1), stride=(1, 1))
              (squeeze_activation): ReLU(inplace=True)
              (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
              (expand1x1_activation): ReLU(inplace=True)
              (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (expand3x3_activation): ReLU(inplace=True)
            )
            (4): Fire(
              (squeeze): Conv2d(128, 16, kernel_size=(1, 1), stride=(1, 1))
              (squeeze_activation): ReLU(inplace=True)
              (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
              (expand1x1_activation): ReLU(inplace=True)
              (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (expand3x3_activation): ReLU(inplace=True)
            )
            (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
            (6): Fire(
              (squeeze): Conv2d(128, 32, kernel_size=(1, 1), stride=(1, 1))
              (squeeze_activation): ReLU(inplace=True)
              (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
              (expand1x1_activation): ReLU(inplace=True)
              (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (expand3x3_activation): ReLU(inplace=True)
            )
            (7): Fire(
              (squeeze): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1))
              (squeeze_activation): ReLU(inplace=True)
              (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
              (expand1x1_activation): ReLU(inplace=True)
              (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (expand3x3_activation): ReLU(inplace=True)
            )
            (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
            (9): Fire(
              (squeeze): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1))
              (squeeze_activation): ReLU(inplace=True)
              (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
              (expand1x1_activation): ReLU(inplace=True)
```

```
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (10): Fire(
      (squeeze): Conv2d(384, 48, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (11): Fire(
      (squeeze): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (12): Fire(
      (squeeze): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Conv2d(512, 1000, kernel_size=(1, 1), stride=(1, 1))
    (2): ReLU(inplace=True)
    (3): AdaptiveAvgPool2d(output_size=(1, 1))
  )
)
```

In [9]:
```
def show_saliency_maps(X, y, model):
    # Convert X and y from numpy arrays to Torch Tensors
    X_tensor = torch.cat([preprocess(Image.fromarray(x)) for x in X], dim=0)
    y_tensor = torch.LongTensor(y)

    # Compute saliency maps for images in X
    saliency = compute_saliency_maps(X_tensor, y_tensor, model)

    # Convert the saliency map from Torch Tensor to numpy array and show images
    # and saliency maps together.
    saliency = saliency.numpy()
    N = X.shape[0]
```

```
        for i in range(N):
            plt.subplot(2, N, i + 1)
            plt.imshow(X[i])
            plt.axis('off')
            plt.title(class_names[y[i]])
            plt.subplot(2, N, N + i + 1)
            #print(type(np.transpose(saliency[i], (1,2,0))))
            plt.imshow(saliency[i], cmap=plt.cm.hot)
            plt.axis('off')
            plt.gcf().set_size_inches(12, 5)
        plt.show()

    for i in range(2): # range(5) pour tester toutes les images
        show_saliency_maps(X[5*i:5*i+5], y[5*i:5*i+5], model)
```






```
In [10]: show_saliency_maps(X[1:1+1], y[1:1+1], model)
```

3 Fooling Images

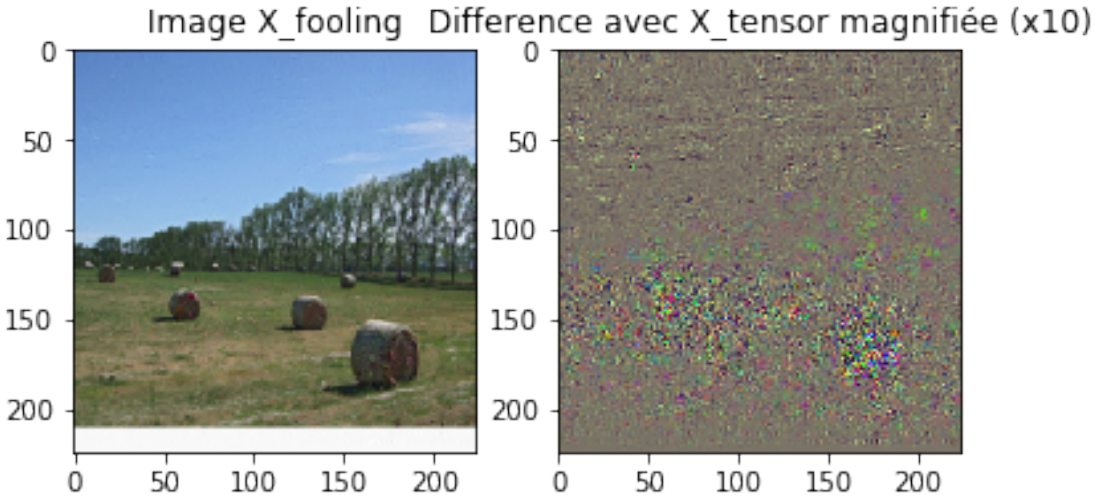
Code pour calculer une image de facon à ce qu'elle soit classée dans une classe `target_y` autre que la classe réelle (en modifiant l'image et pas les poids du réseau).

Les 2 premiers blocs permettent de faire des tests de facon interractive pour écrire votre code. Une fois que votre code semble fonctionner, compléter la fonction dans le 3e bloc et tester sur differentes images avec le 4e bloc.

```
In [11]: # Init du test
         X_tensor = torch.Tensor(preprocess(Image.fromarray(X[0])))
         target_y = 6
         X_fooling = X_tensor.clone()
         X_fooling.requires_grad = True
         learning_rate = 1.0

In [12]: t = 0
         while True:
             yo = model.forward(X_fooling)
             yy = yo.argmax(dim=1)[0].item()
             if yy == target_y:
                 break
```

```
            yo[0,target_y].backward()
            with torch.no_grad():
                X_fooling.data += learning_rate*X_fooling.grad.data
            print(t,end='->')
            t+=1

        # Affichage de l'image X_fooling et ses modifs
        plt.subplot(1, 2, 1)
        plt.imshow(np.asarray(deprocess(X_fooling.clone())).astype(np.uint8))
        plt.title("Image X_fooling")
        plt.subplot(1, 2, 2)
        plt.imshow(np.asarray(deprocess(10* (X_fooling - X_tensor), should_rescale=False)))
        plt.title("Difference avec X_tensor magnifiée (x10)")
        plt.show()
```

```
0->1->2->
```

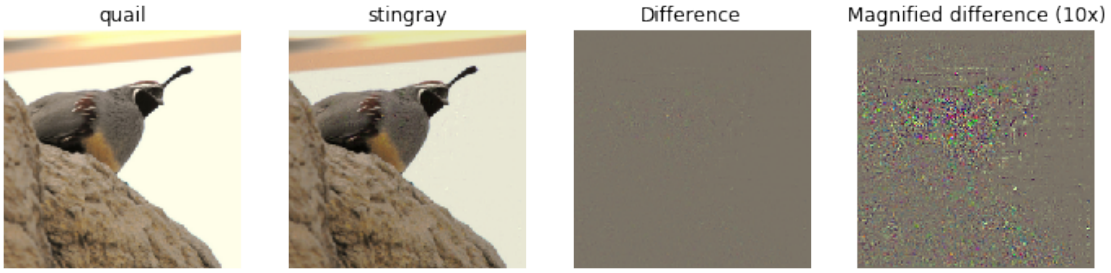



```
In [13]: def make_fooling_image(X, target_y, model, nb_iterations=100):
            """
            Generate a fooling image that is close to X, but that the model classifies
            as target_y.

            Inputs:
            - X: Input image; Tensor of shape (1, 3, 224, 224)
            - target_y: An integer in the range [0, 1000)
            - model: A pretrained CNN

            Returns:
            - X_fooling: An image that is close to X, but that is classifed as target_y
            by the model.
```

```
    """
    # Initialize our fooling image to the input image, and wrap it in a Variable.
    X_fooling = X.clone()
    X_fooling.requires_grad = True

    learning_rate = 0.1
    ################################################################################
    # TODO: Generate a fooling image X_fooling that the model will classify as   #
    # the class target_y. You should perform gradient ascent on the score of the #
    # target class, stopping when the model is fooled.                           #
    # When computing an update step, first normalize the gradient:               #
    #   dX = learning_rate * g / ||g||_2                                         #
    #                                                                            #
    # You should write a training loop.                                          #
    #                                                                            #
    # HINT: For most examples, you should be able to generate a fooling image    #
    # in fewer than 100 iterations of gradient ascent.                           #
    # You can print your progress over iterations to check your algorithm.       #
    ################################################################################

    while True:
        yo = model.forward(X_fooling)
        yy = yo.argmax(dim=1)[0].item()
        if yy == target_y:
            break
        yo[0,target_y].backward()
        with torch.no_grad():
            X_fooling.data += learning_rate*X_fooling.grad.data

    ################################################################################
    #                              END OF YOUR CODE                                #
    ################################################################################
    return X_fooling
```

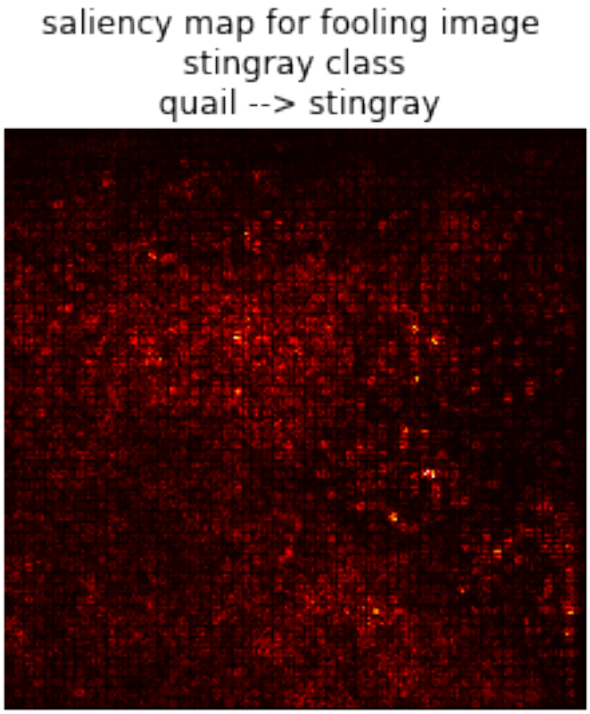
In [14]:
```
# Indice de l'image a modifier et de la classe cible
idx = 1
target_y = 6

# Preparation du tenseur X et sa version "fooling"
X_tensor = torch.cat([preprocess(Image.fromarray(x)) for x in X], dim=0)
X_fooling = make_fooling_image(X_tensor[idx:idx+1], target_y, model)

# verification de la classe predite
scores = model(X_fooling)
assert target_y == scores.data.max(1)[1][0], 'The model is not fooled!'

# Affichage
X_fooling_np = deprocess(X_fooling.clone())
```

```
X_fooling_np = np.asarray(X_fooling_np).astype(np.uint8)

plt.subplot(1, 4, 1)
plt.imshow(X[idx])
plt.title(class_names[y[idx]])
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(X_fooling_np)
plt.title(class_names[target_y])
plt.axis('off')

plt.subplot(1, 4, 3)
X_pre = preprocess(Image.fromarray(X[idx]))
diff = np.asarray(deprocess(X_fooling - X_pre, should_rescale=False))
plt.imshow(diff)
plt.title('Difference')
plt.axis('off')

plt.subplot(1, 4, 4)
diff = np.asarray(deprocess(10 * (X_fooling - X_pre), should_rescale=False))
plt.imshow(diff)
plt.title('Magnified difference (10x)')
plt.axis('off')

plt.gcf().set_size_inches(12, 5)
plt.show()
```




```
In [15]: X_fooling.shape, np.array([target_y])[None].shape

Out[15]: (torch.Size([1, 3, 224, 224]), (1, 1))

In [16]: type(compute_saliency_maps(X_fooling, np.array([target_y]), model))

Out[16]: torch.Tensor

In [17]: plt.imshow(compute_saliency_maps(X_fooling, np.array([target_y]), model)[0] , cmap=plt
         plt.title("saliency map for fooling image \n stingray class \n quail --> stingray")
         plt.axis("off")
```

```
Out[17]: (-0.5, 223.5, 223.5, -0.5)
```



4 Class visualization

Code permettant de calculer une image maximisant le score d'une classe, sujet à un certain nombre de régularisations.

```
In [18]: def create_class_visualization(target_y, model, dtype, init_img=None, l2_reg=1e-3, lea
                                       num_iterations=200, blur_every=10, max_jitter=16, show_
             """
             Generate an image to maximize the score of target_y under a pretrained model.

             Inputs:
             - target_y: Integer in the range [0, 1000) giving the index of the class
             - model: A pretrained CNN that will be used to generate the image
             - dtype: Torch datatype to use for computations

             Keyword arguments:
             - init_img: Initial image to use (if None, will be random)
             - l2_reg: Strength of L2 regularization on the image
             - learning_rate: How big of a step to take
             - num_iterations: How many iterations to use
             - blur_every: How often to blur the image as an implicit regularizer
```

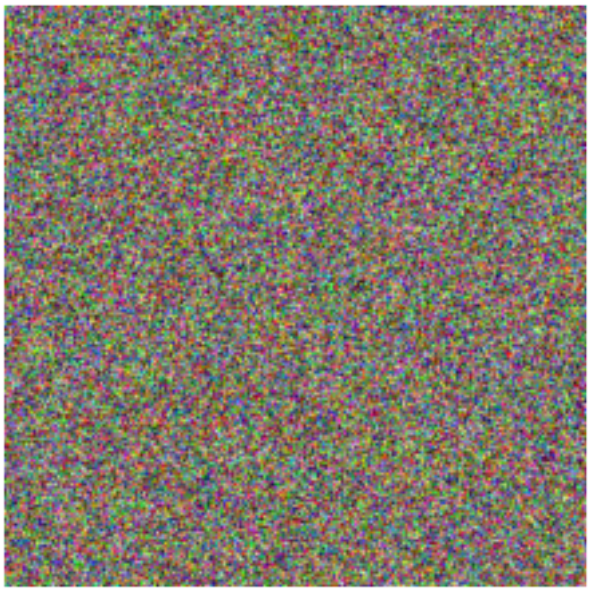
```
    - max_jitter: How much to gjitter the image as an implicit regularizer
    - show_every: How often to show the intermediate result
    """
    model.type(dtype)

    # Randomly initialize the image as a PyTorch Tensor:
    if init_img is None:
        img = torch.randn(1, 3, 224, 224).mul_(1.0).type(dtype)
    else:
        img = init_img.clone().mul_(1.0).type(dtype)
    img.requires_grad = True

    for t in range(num_iterations):
        # Randomly jitter the image a bit; this gives slightly nicer results
        ox, oy = random.randint(0, max_jitter), random.randint(0, max_jitter)
        img.data.copy_(jitter(img.data, ox, oy))

        ########################################################################
        # TODO: Use the model to compute the gradient of the score for the     #
        # class target_y with respect to the pixels of the image, and make a   #
        # gradient step on the image using the learning rate. Don't forget the #
        # L2 regularization term!                                              #
        # Be very careful about the signs of elements in your code.            #
        ########################################################################

        y_hat = model.forward(img)
        #print(y_hat.argmax(dim=1)[0].item(), class_names[y_hat.argmax(dim=1)[0].item
        loss = y_hat[0, target_y] - l2_reg * torch.norm(img,2)
        loss.backward()

        with torch.no_grad():
            img += learning_rate*img.grad

        ########################################################################
        #                             END OF YOUR CODE                         #
        ########################################################################

        # Undo the random jitter
        img.data.copy_(jitter(img.data, -ox, -oy))

        # As regularizer, clamp and periodically blur the image
        for c in range(3):
            lo = float(-SQUEEZENET_MEAN[c] / SQUEEZENET_STD[c])
            hi = float((1.0 - SQUEEZENET_MEAN[c]) / SQUEEZENET_STD[c])
            img.data[:, c].clamp_(min=lo, max=hi)
        if t % blur_every == 0:
            img.data = blur_image(img.data.cpu().numpy(), sigma=0.5)
```

```
            # Periodically show the image
            if t == 0 or (t + 1) % show_every == 0 or t == num_iterations - 1:
                plt.imshow(deprocess(img.clone().cpu()))
                class_name = class_names[target_y]
                cn = class_names[y_hat.argmax(dim=1)[0].item()]
                plt.title('y_hat:%s\ny:%s\nIteration %d / %d' % (cn,class_name, t + 1, nu
                plt.gcf().set_size_inches(4, 4)
                plt.axis('off')
                plt.show()

        return deprocess(img.cpu())
```

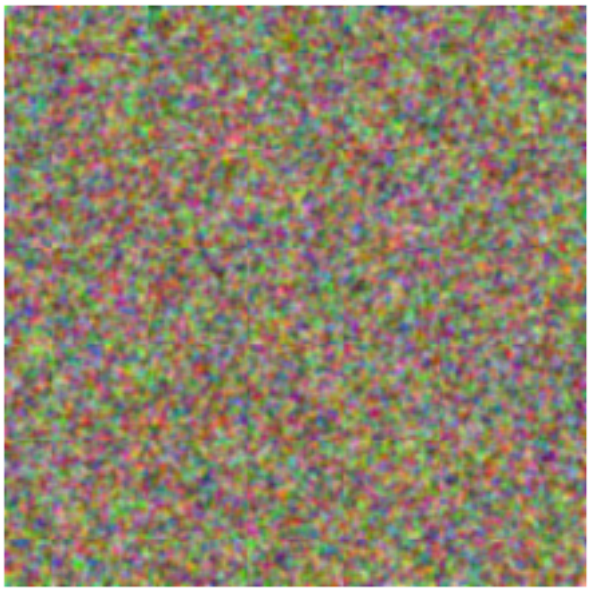
Test avec diverses classes en partant d'un bruit aléatoire :

```
In [19]: dtype = torch.FloatTensor
         # dtype = torch.cuda.FloatTensor # Uncomment this to use GPU
         model.type(dtype)

         l2_reg = 1e-3
         learning_rate = 0.1

         num_iterations = 1000

         #target_y = 281 # Tabby cat
         # target_y = 187 # Yorkshire Terrier
         #target_y = 76 # Tarantula
         # target_y = 78 # Tick
         # target_y = 683 # Oboe
         target_y = 366 # Gorilla
         # target_y = 604 # Hourglass
         # target_y = np.random.randint(1000) # Classe aléatoire
         out = create_class_visualization(target_y, model, dtype, show_every=50, num_iteration
```

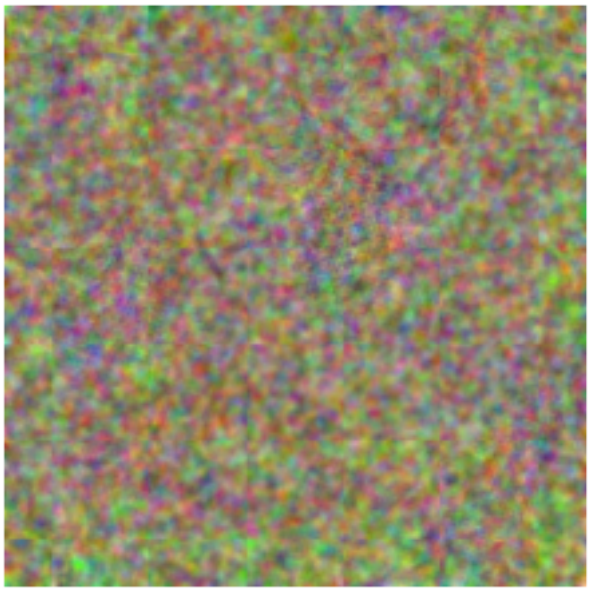
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 1 / 1000



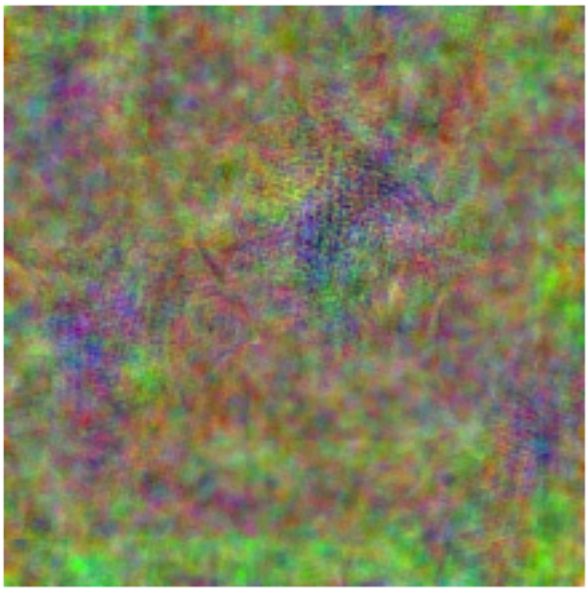
y_hat:skunk, polecat, wood pussy
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



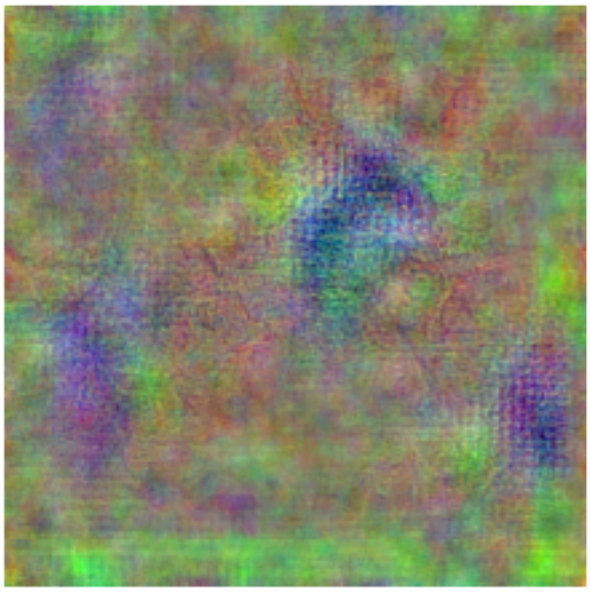
y_hat:bustard
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



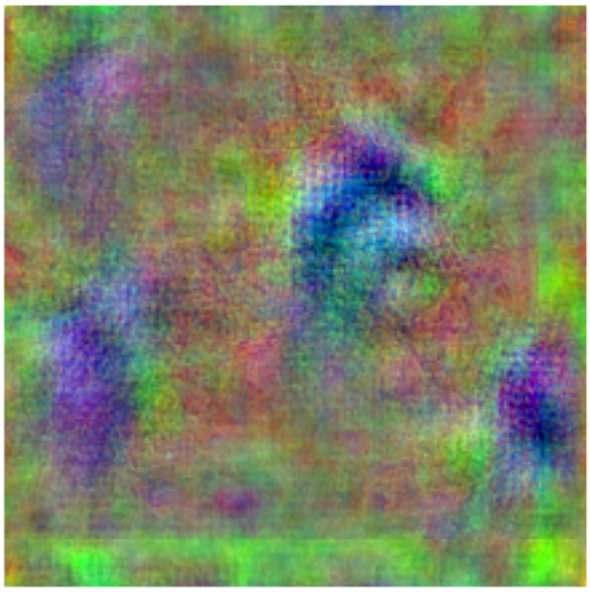
y_hat:grey fox, gray fox, Urocyon cinereoargenteus
y:gorilla, Gorilla gorilla
Iteration 150 / 1000



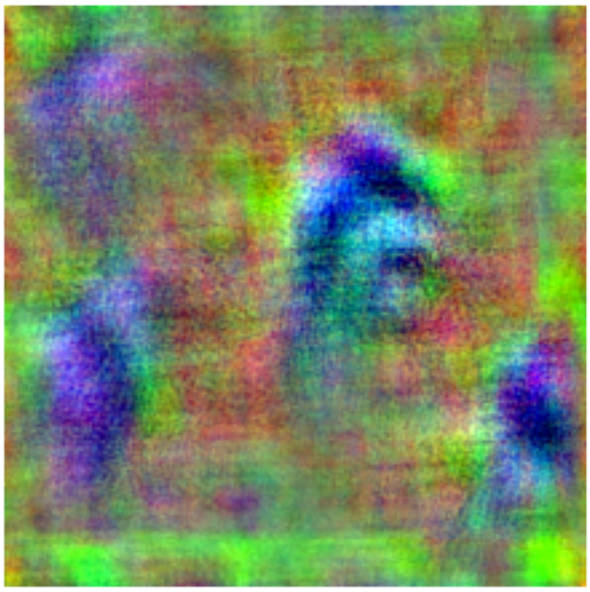
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 200 / 1000



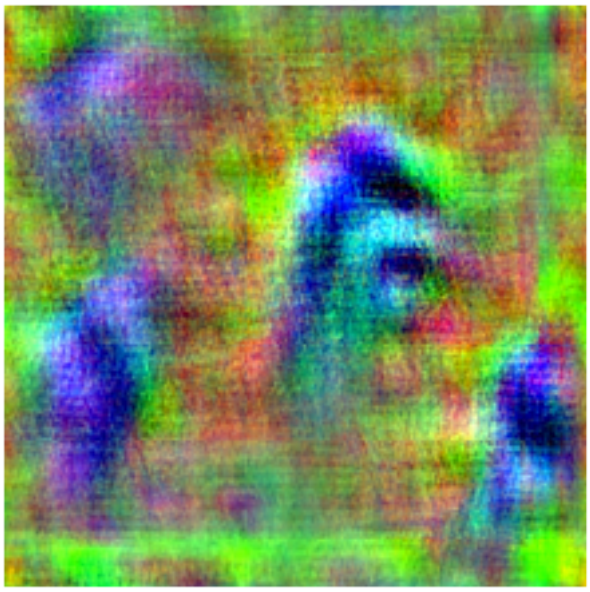
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



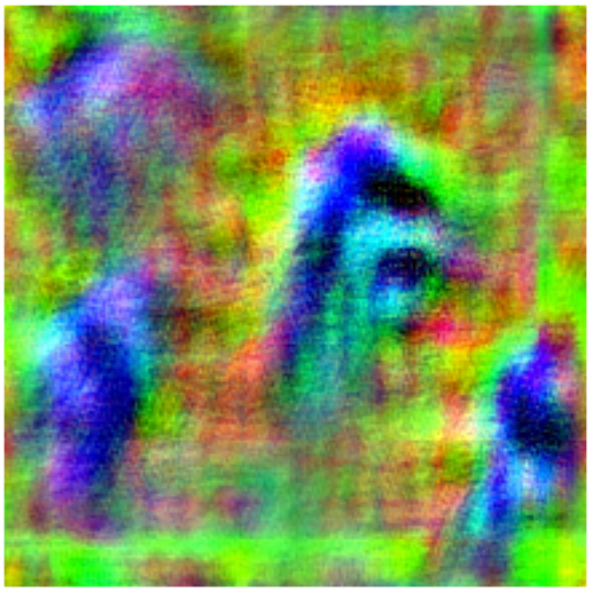
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



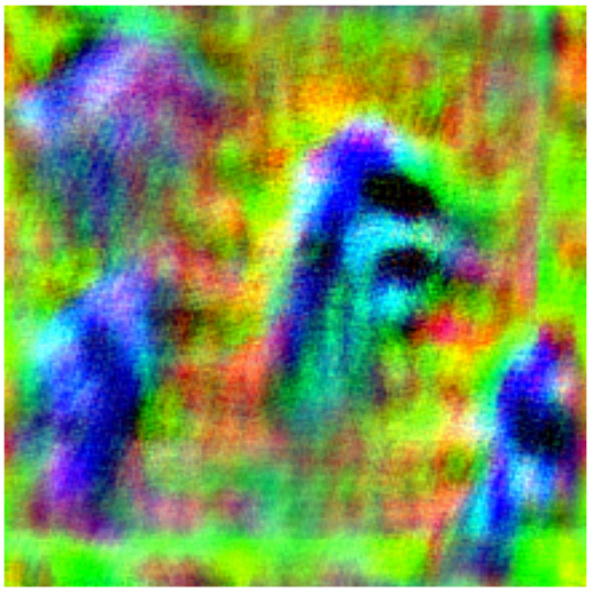
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 350 / 1000



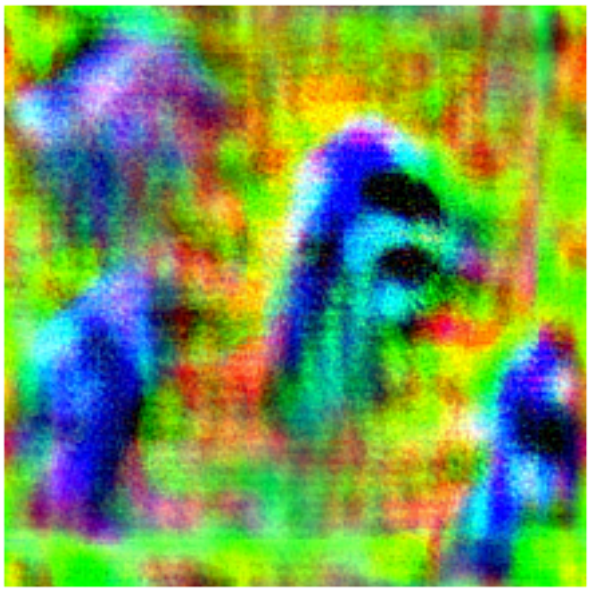
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 400 / 1000



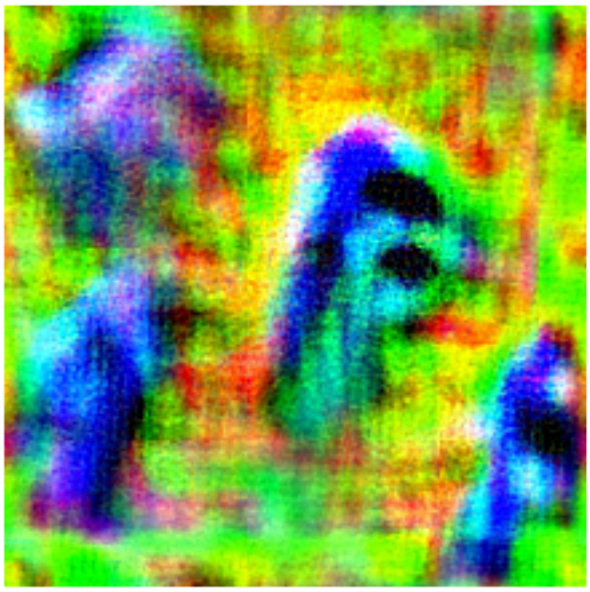
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 450 / 1000



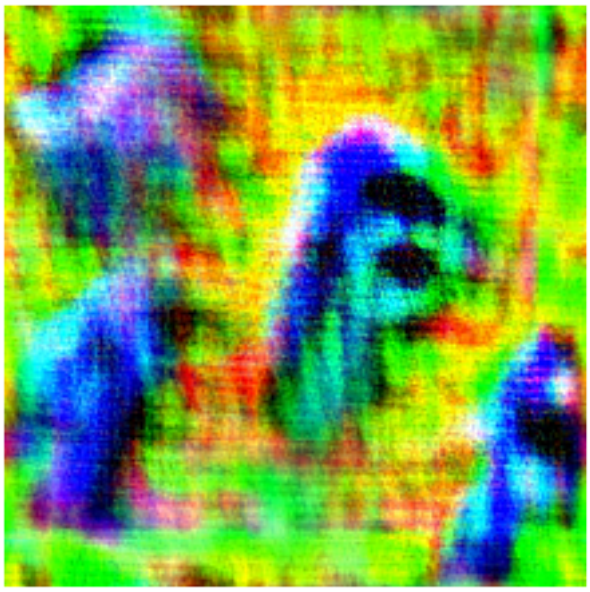
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



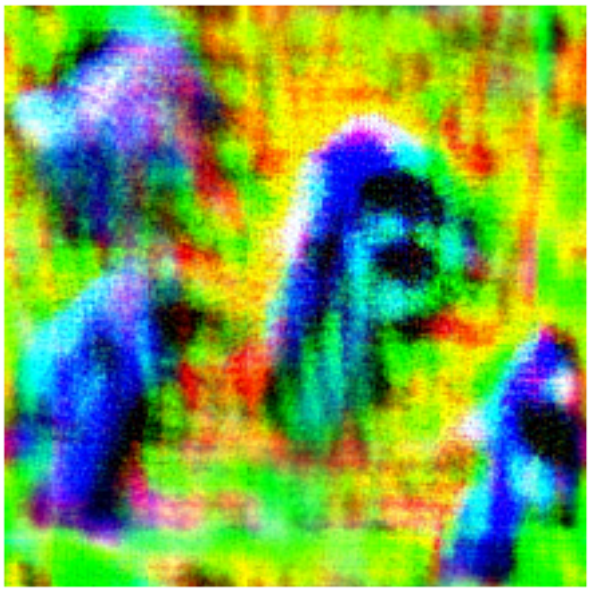
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



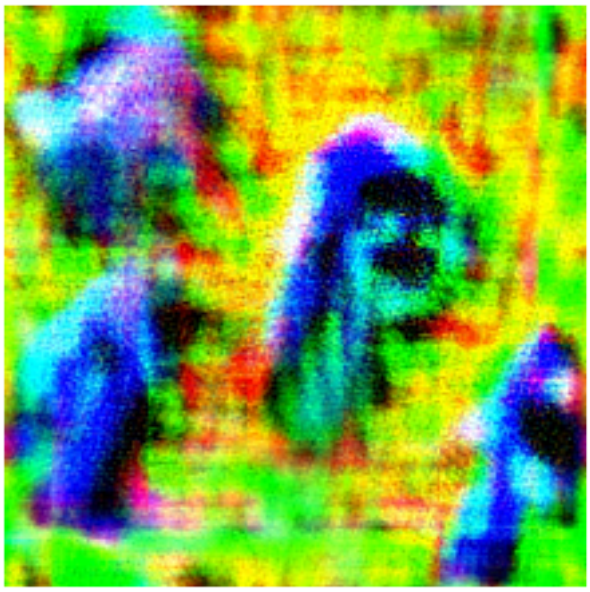
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



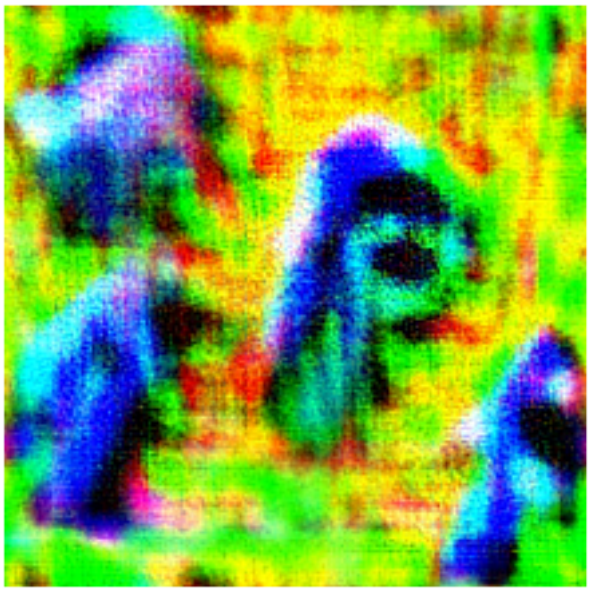
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



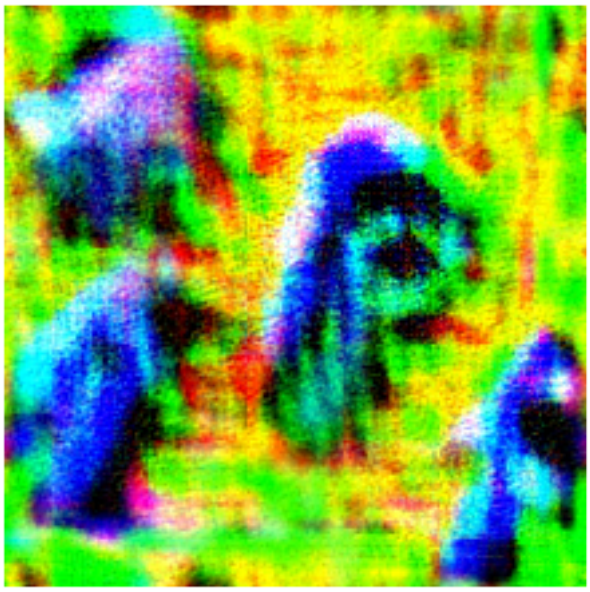
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



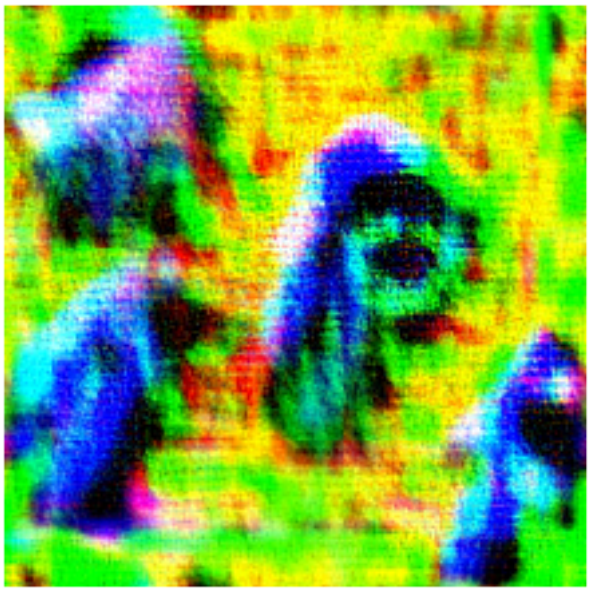
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 750 / 1000



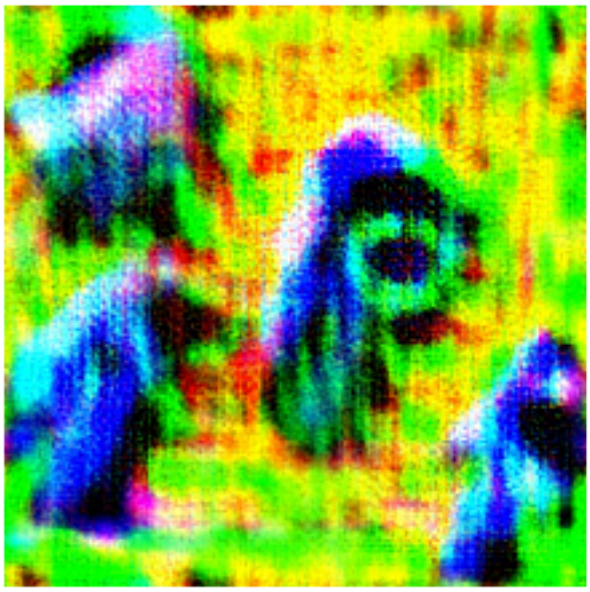
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 800 / 1000



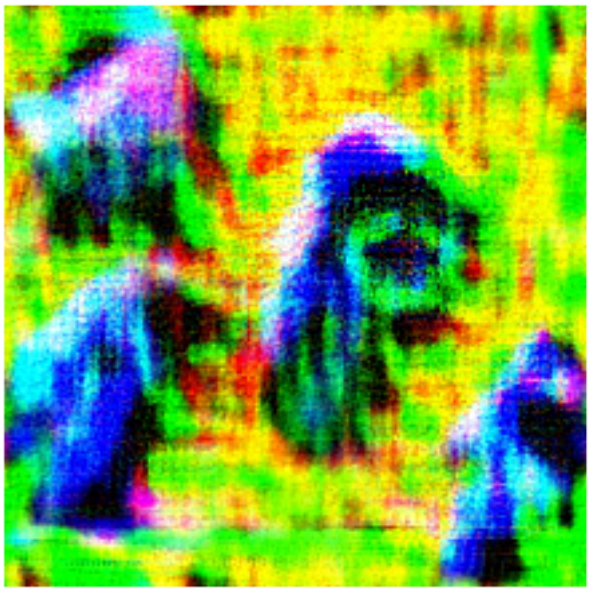
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 850 / 1000

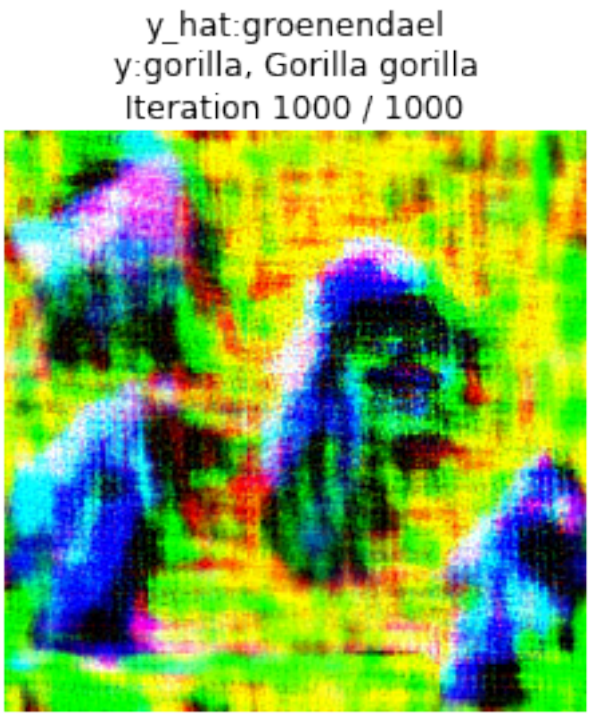


y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



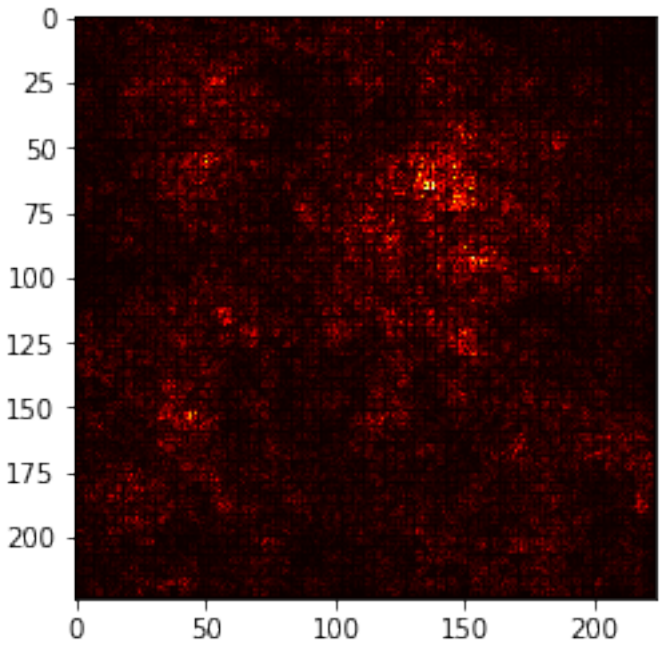
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 950 / 1000





In [20]: plt.imshow(compute_saliency_maps(preprocess(out), np.array([target_y]), model)[0] , cm

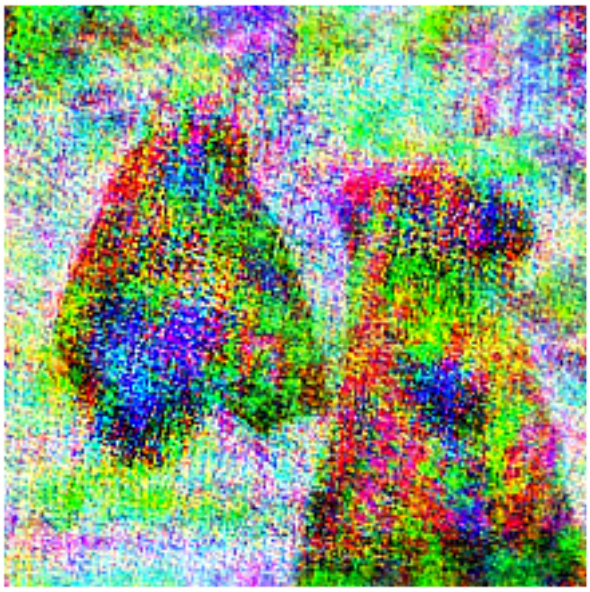
Out[20]: <matplotlib.image.AxesImage at 0x7fc8b44b5b00>



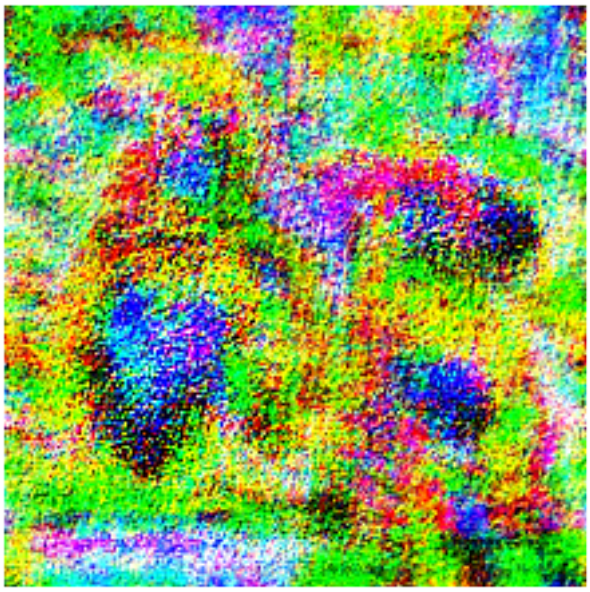
Test en partant d'une image d'ImageNet :

In [21]:
```
# Init du test
img_ind = 4


target_y = y[img_ind]

#target_y = 281 # Tabby cat


l2_reg = 1e-2
learning_rate = 0.1



X_tensor = torch.Tensor(preprocess(Image.fromarray(X[img_ind])))
out = create_class_visualization(target_y, model, dtype, init_img=X_tensor, show_every
```

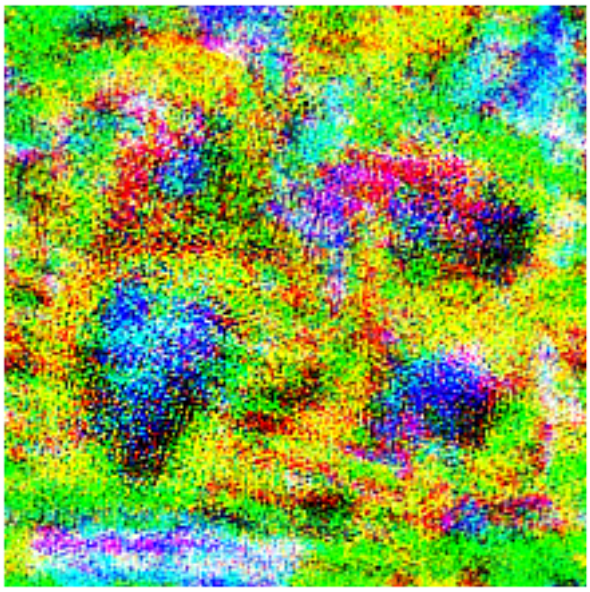
y_hat:coral reef
y:brown bear, bruin, Ursus arctos
Iteration 50 / 1000



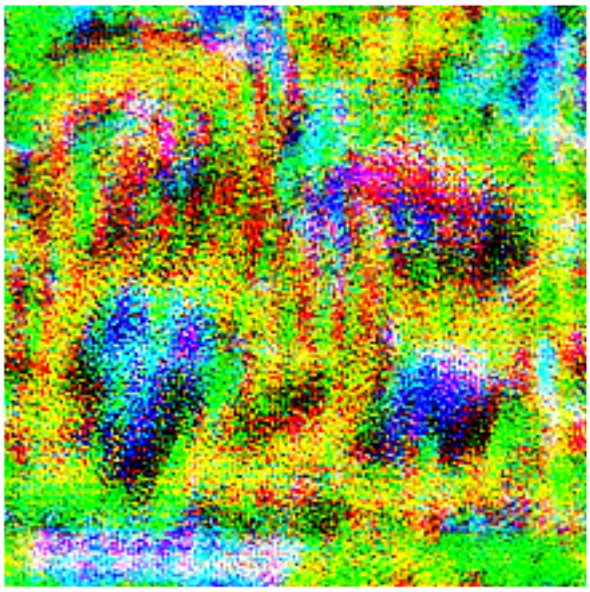
y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 100 / 1000



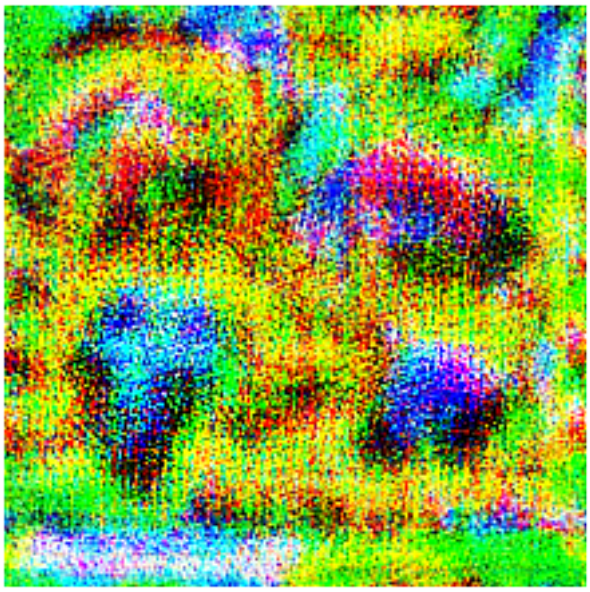
y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 150 / 1000



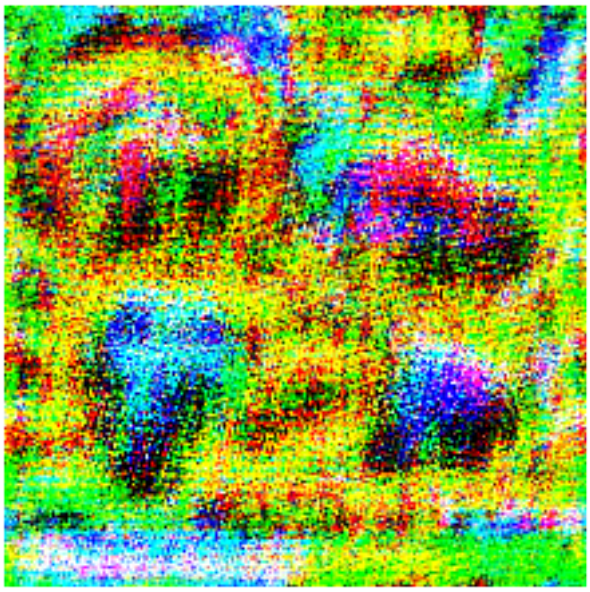
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 200 / 1000



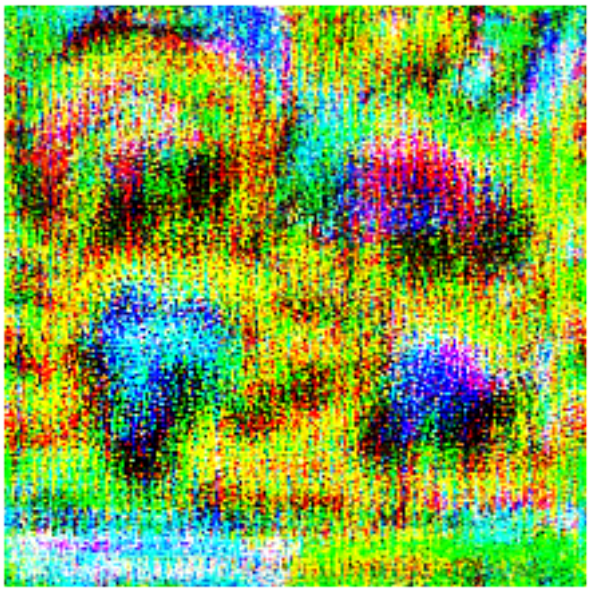
y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 250 / 1000



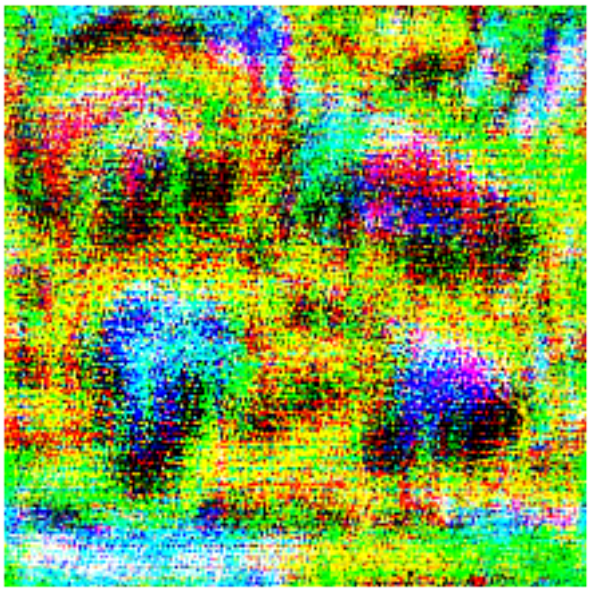
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 300 / 1000



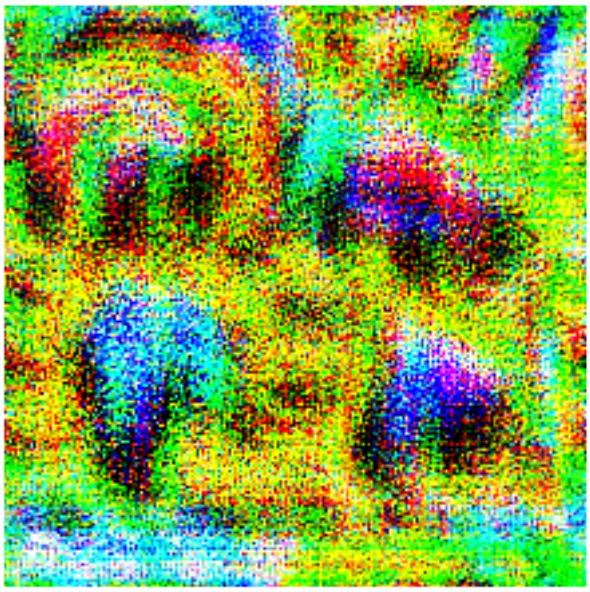
y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 350 / 1000



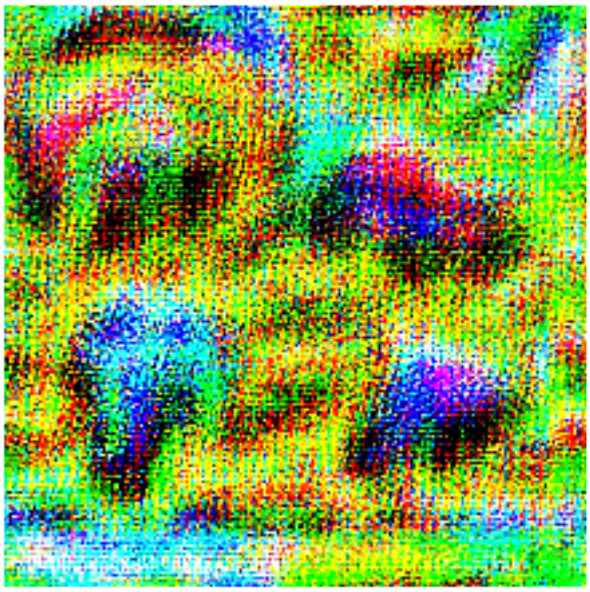
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 400 / 1000



y_hat:coral reef
y:brown bear, bruin, Ursus arctos
Iteration 450 / 1000

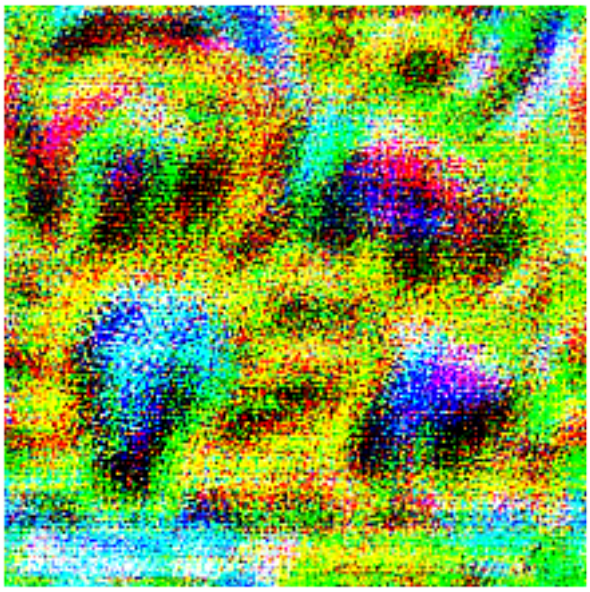


y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 500 / 1000



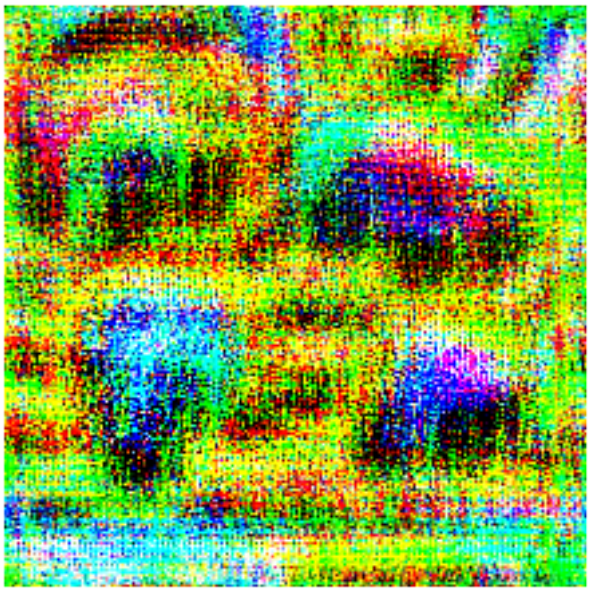
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 550 / 1000

y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 600 / 1000

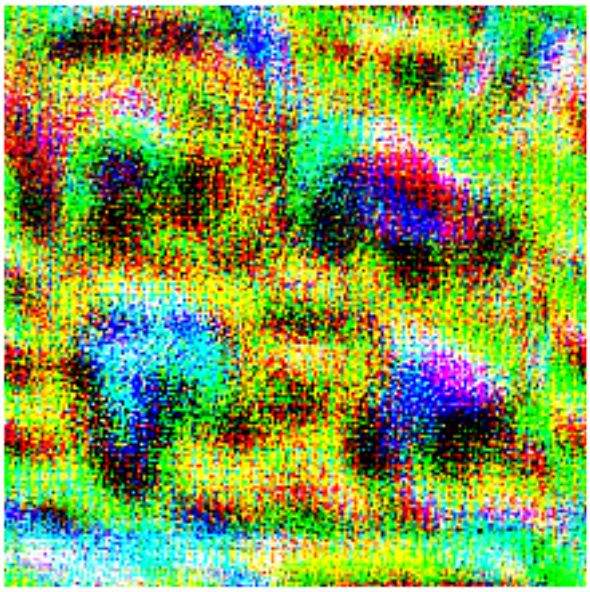


y_hat:brown bear, bruin, Ursus arctos
y:brown bear, bruin, Ursus arctos
Iteration 650 / 1000

y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 700 / 1000



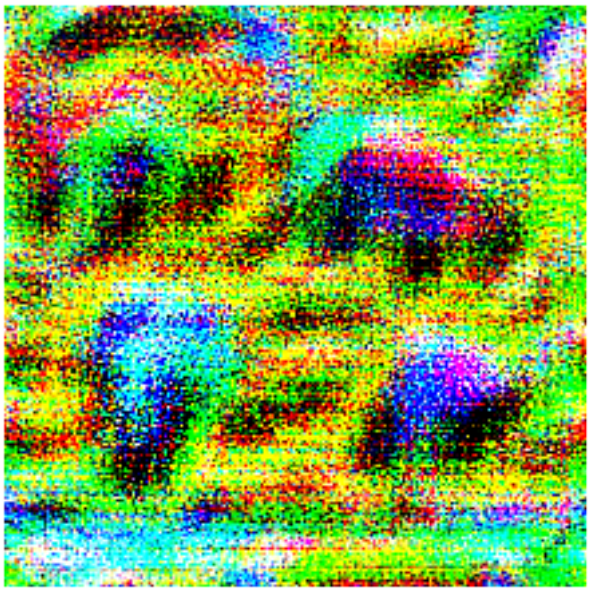
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 750 / 1000



y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 800 / 1000

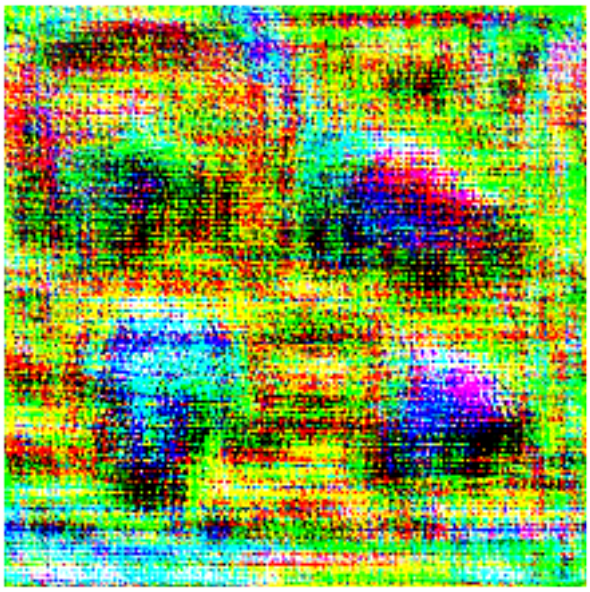
y_hat:brown bear, bruin, Ursus arctos
y:brown bear, bruin, Ursus arctos
Iteration 850 / 1000

y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 900 / 1000



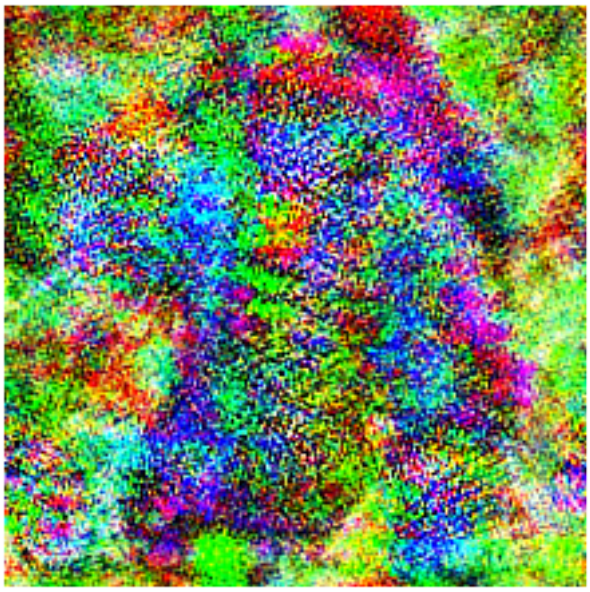
y_hat:peacock
y:brown bear, bruin, Ursus arctos
Iteration 950 / 1000

y_hat:jigsaw puzzle
y:brown bear, bruin, Ursus arctos
Iteration 1000 / 1000

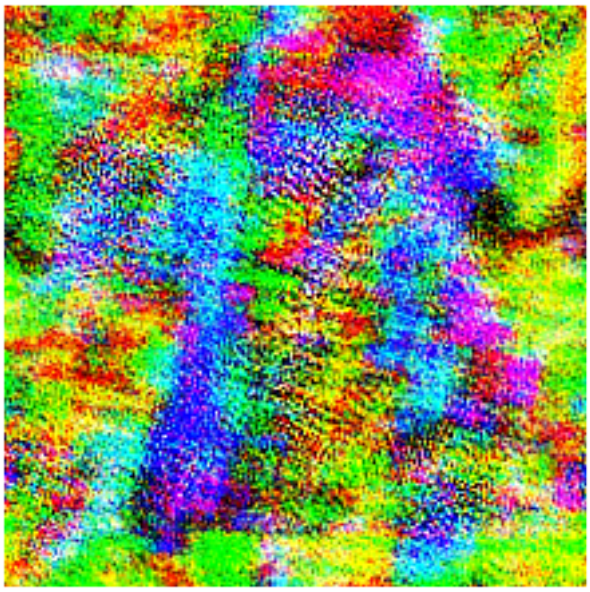


In [22]:
```
# Init du test
img_ind = 7


target_y = y[img_ind]

#target_y = 281 # Tabby cat


l2_reg = 1e-2
learning_rate = 0.1



X_tensor = torch.Tensor(preprocess(Image.fromarray(X[img_ind])))
out = create_class_visualization(target_y, model, dtype, init_img=X_tensor, show_every
```

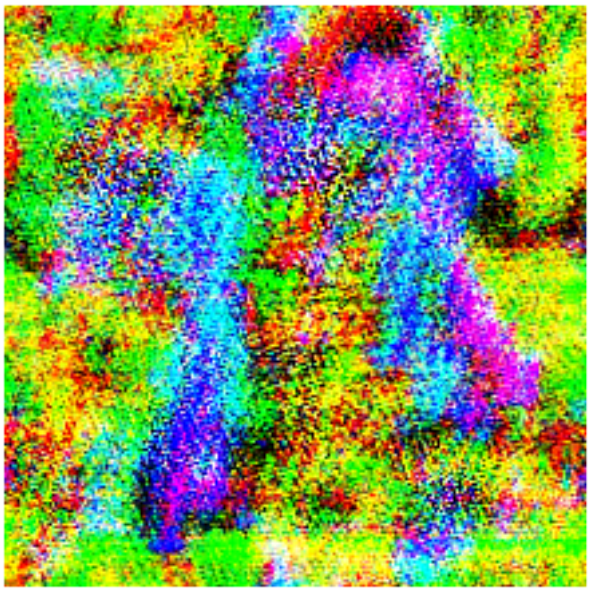
y_hat:jigsaw puzzle
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



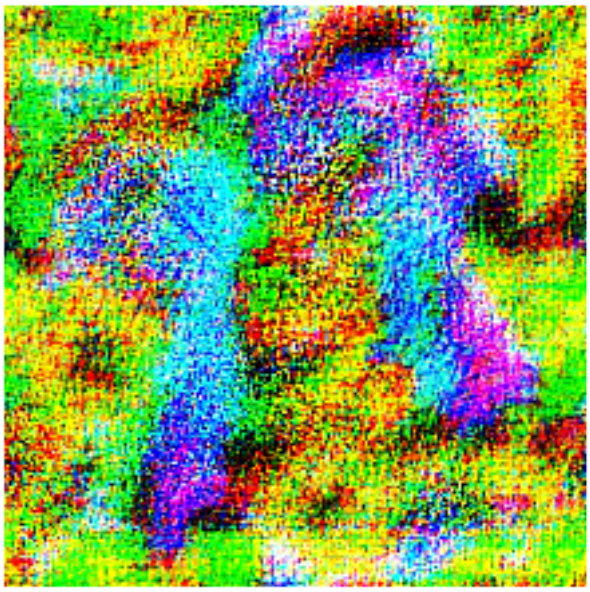
y_hat:sock
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



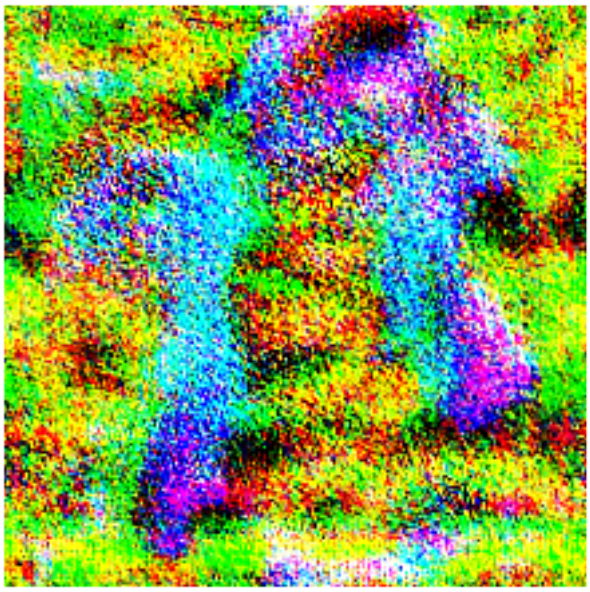
y_hat:starfish, sea star
y:gorilla, Gorilla gorilla
Iteration 150 / 1000



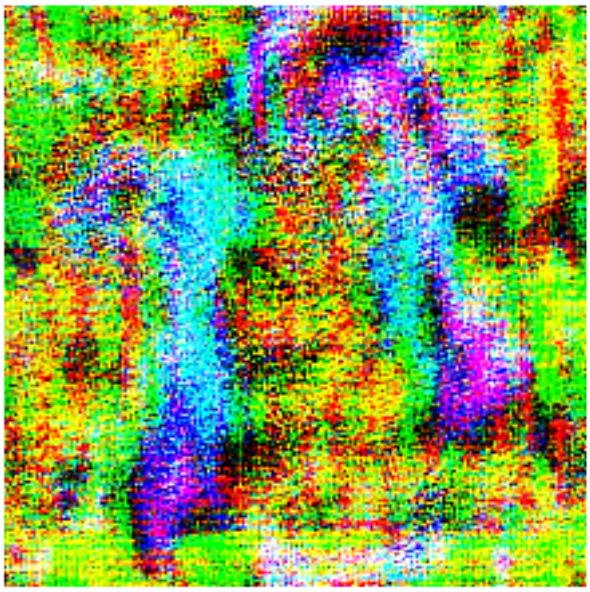
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 200 / 1000



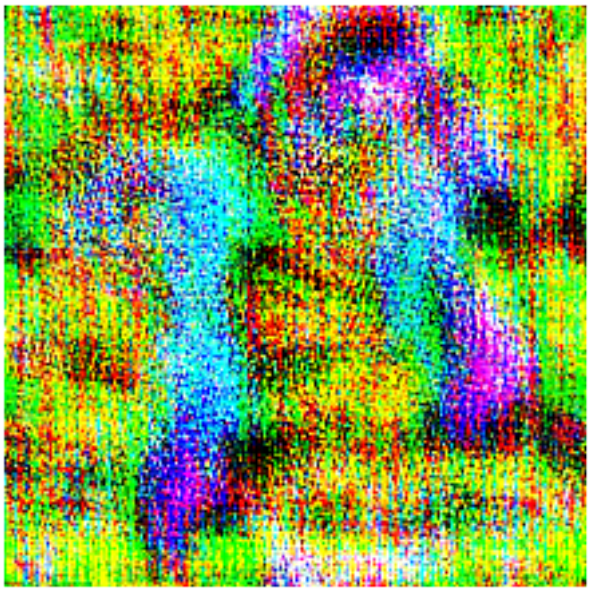
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



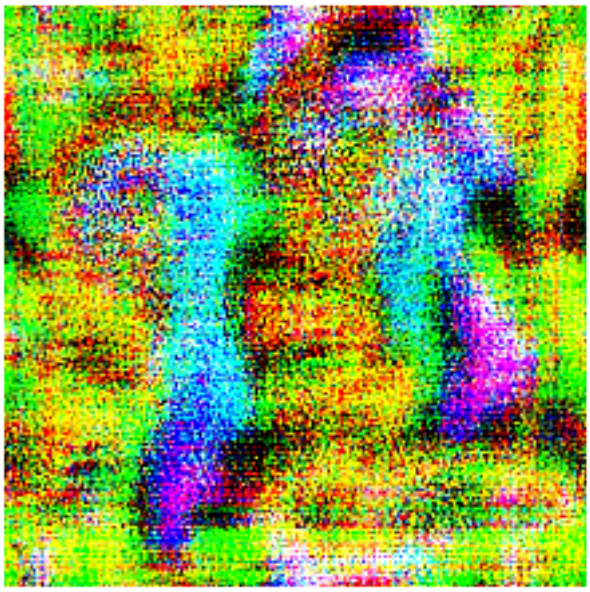
y_hat:sock
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 350 / 1000

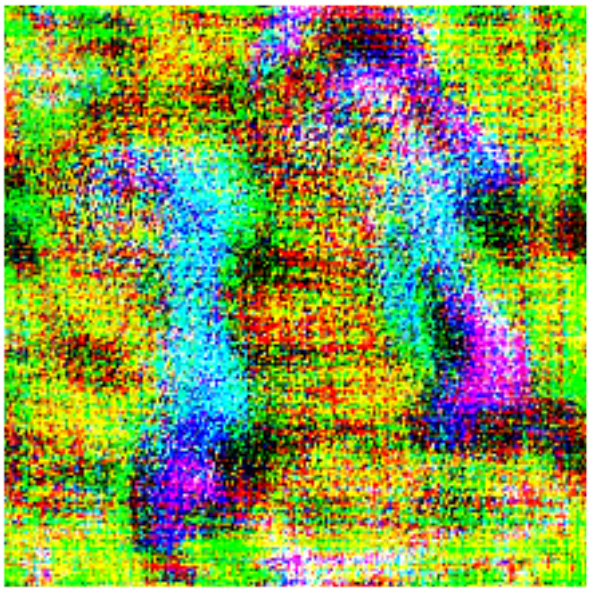


y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 400 / 1000

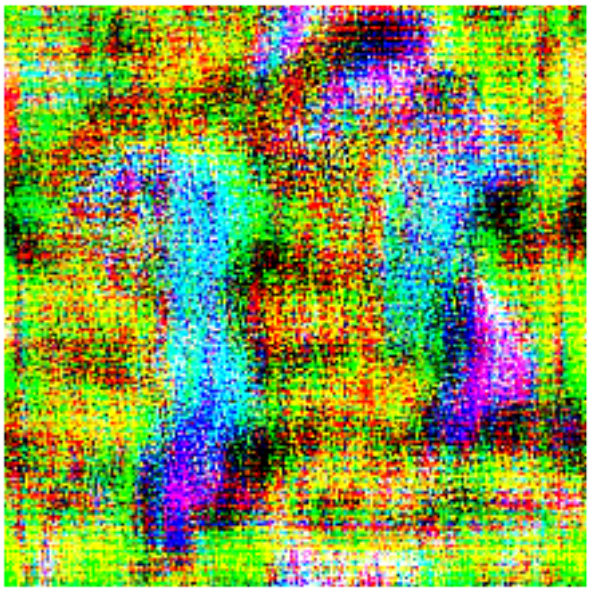


y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 450 / 1000

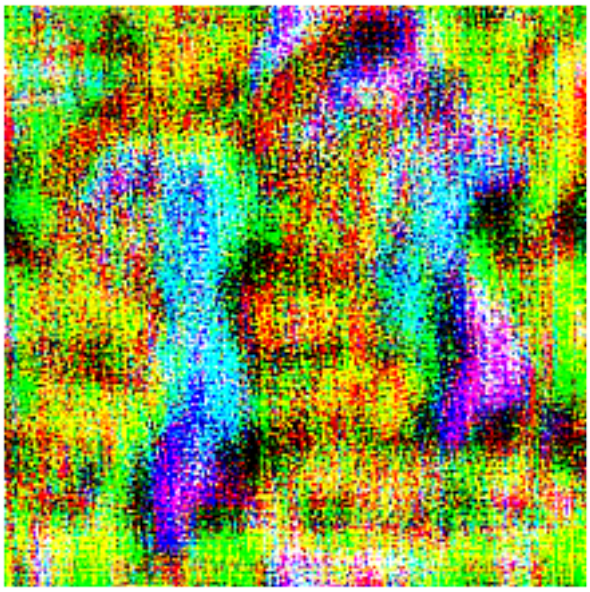
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



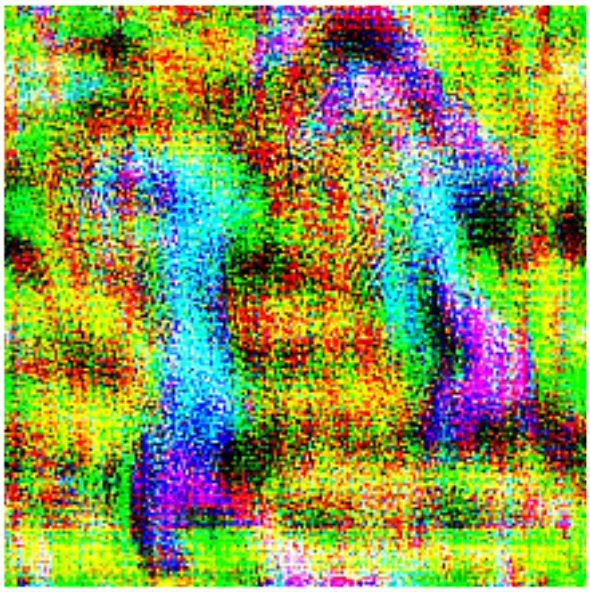
y_hat:jigsaw puzzle
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



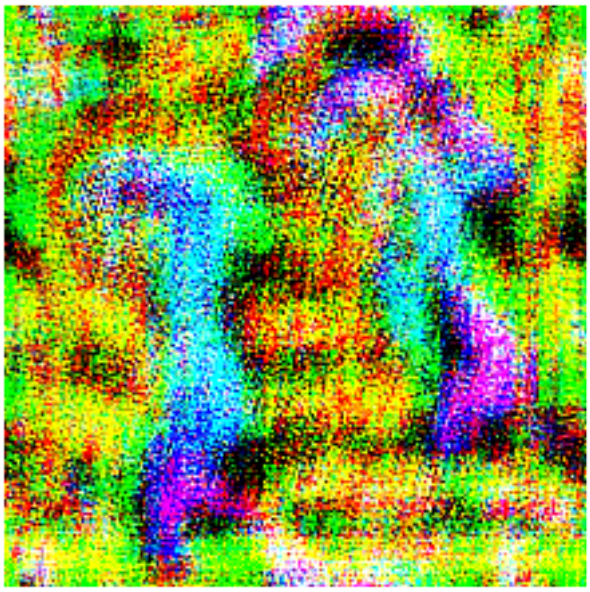
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



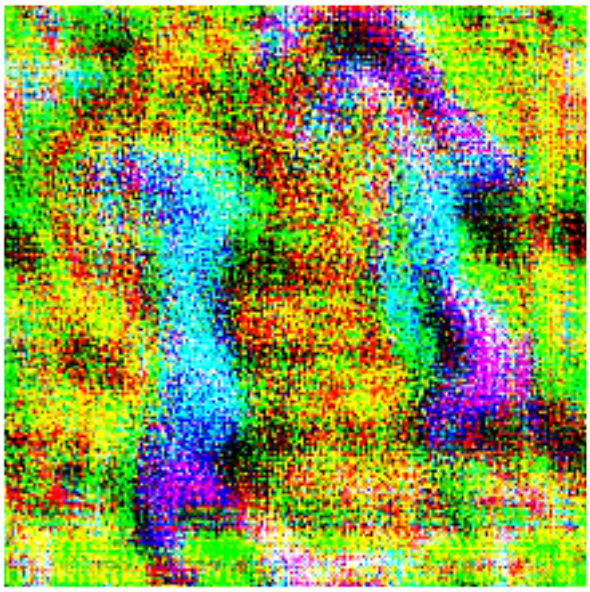
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



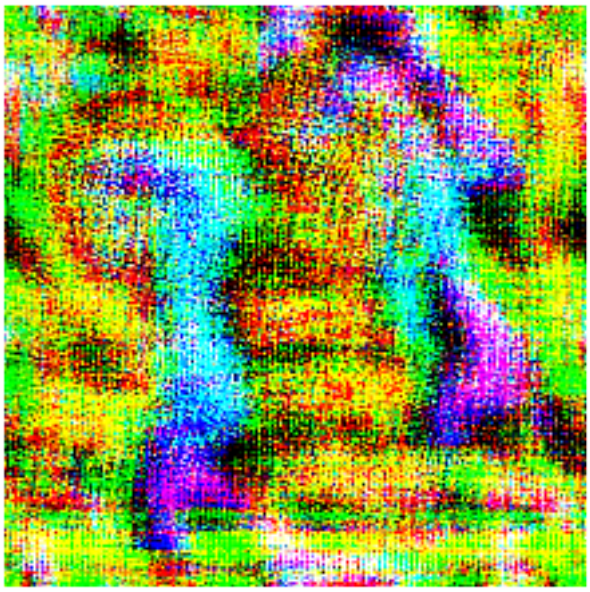
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 750 / 1000

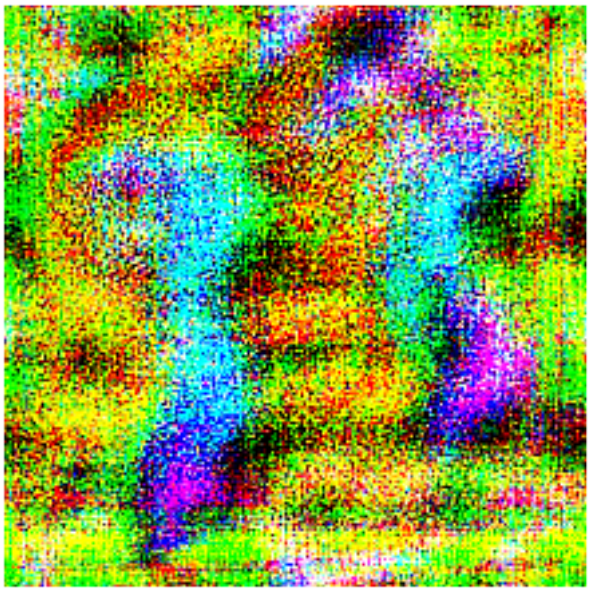


y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 800 / 1000

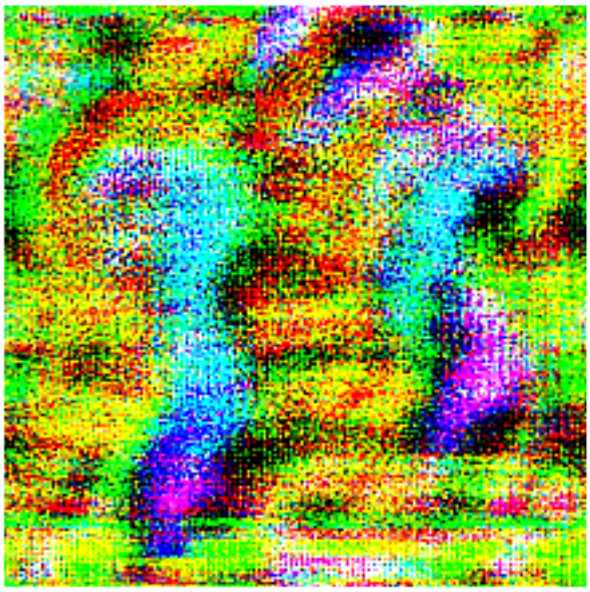


y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 850 / 1000

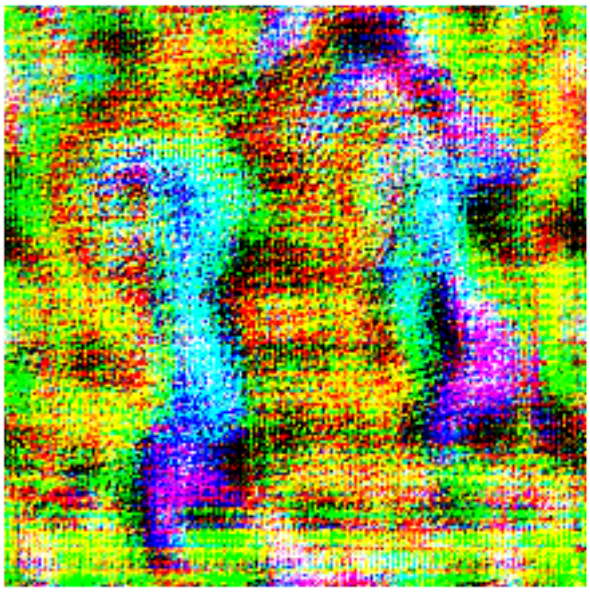
y_hat:coral reef
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 950 / 1000

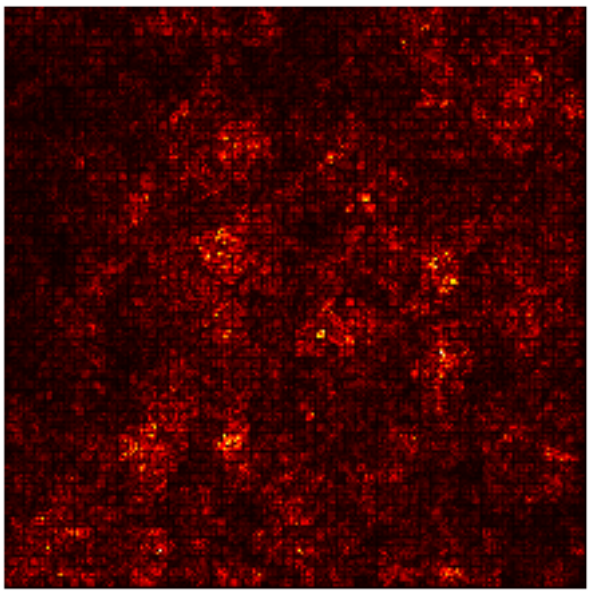


y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 1000 / 1000



In [23]:
```
plt.imshow(compute_saliency_maps(preprocess(out), np.array([target_y]), model)[0] , cm
plt.axis("off")
```

Out[23]: (-0.5, 223.5, 223.5, -0.5)

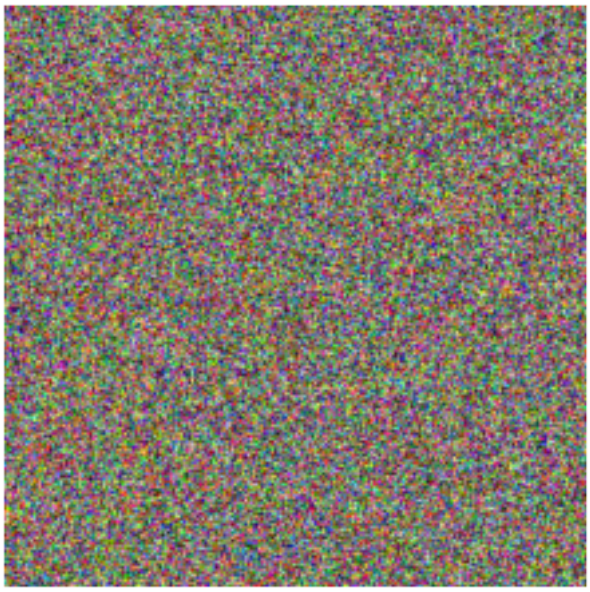


In [24]:
```
class_names[target_y]
```

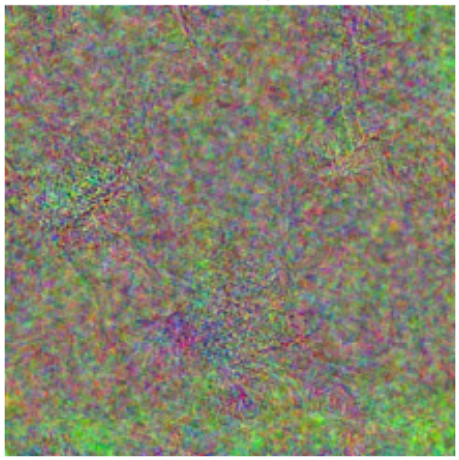
Out[24]: 'gorilla, Gorilla gorilla'

In [25]:
```
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to use GPU
model.type(dtype)

l2_reg = 1e-3
learning_rate = 1

num_iterations = 1000

#target_y = 281 # Tabby cat
# target_y = 187 # Yorkshire Terrier
#target_y = 76 # Tarantula
# target_y = 78 # Tick
# target_y = 683 # Oboe
target_y = 366 # Gorilla
# target_y = 604 # Hourglass
# target_y = np.random.randint(1000) # Classe aléatoire
out = create_class_visualization(target_y, model, dtype, show_every=50, num_iterations
```

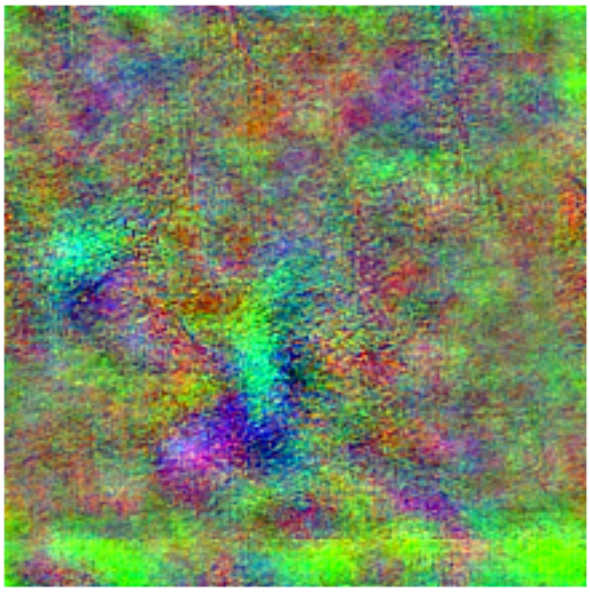
y_hat:poncho
y:gorilla, Gorilla gorilla
Iteration 1 / 1000



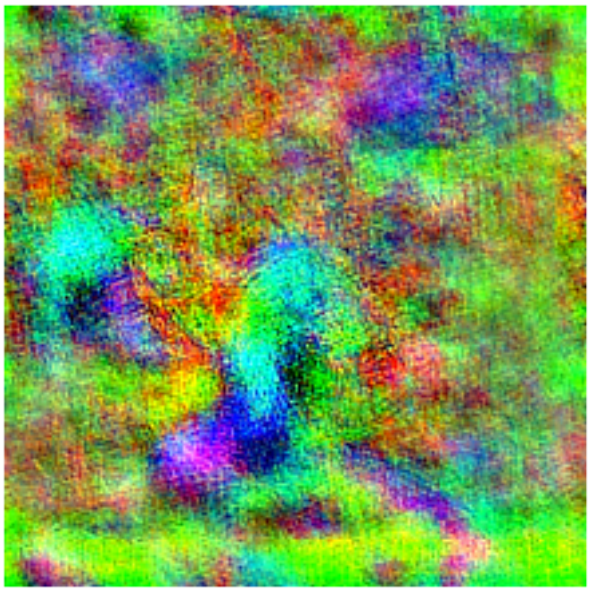
y_hat:Komodo dragon, Komodo lizard, dragon lizard, giant lizard, Varanus komodoensis
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



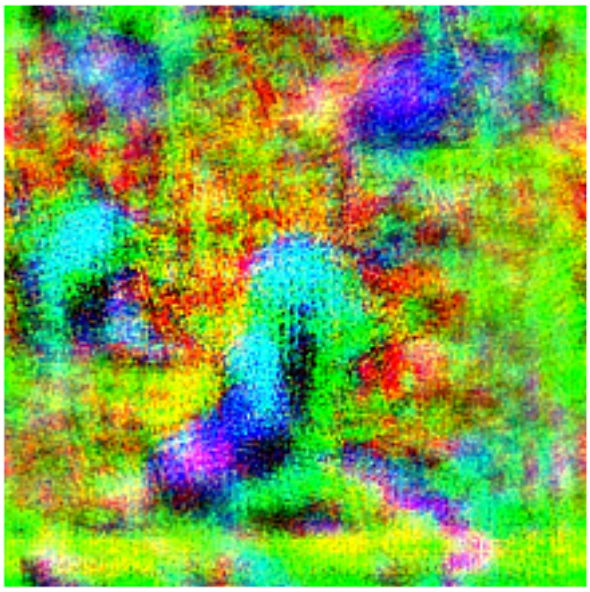
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



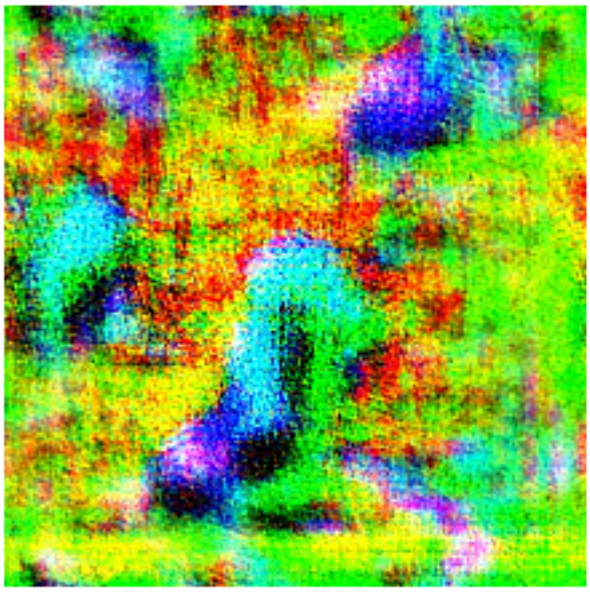
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 150 / 1000



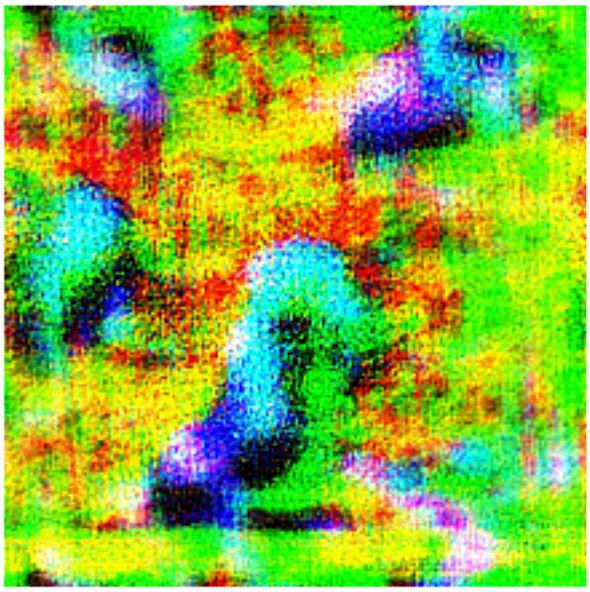
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 200 / 1000



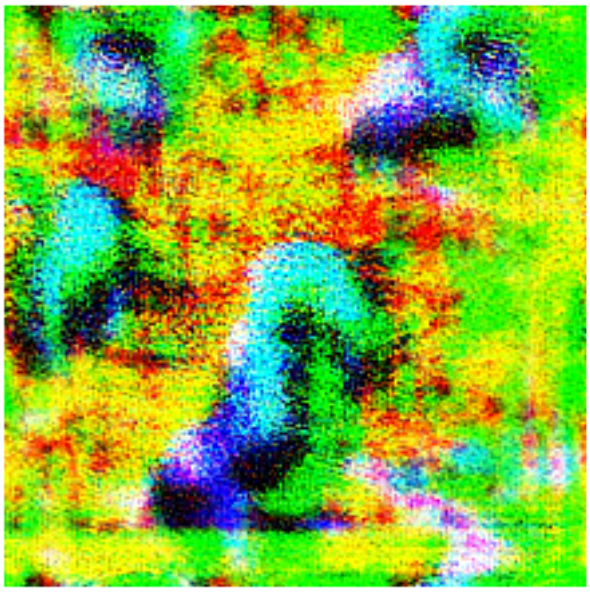
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



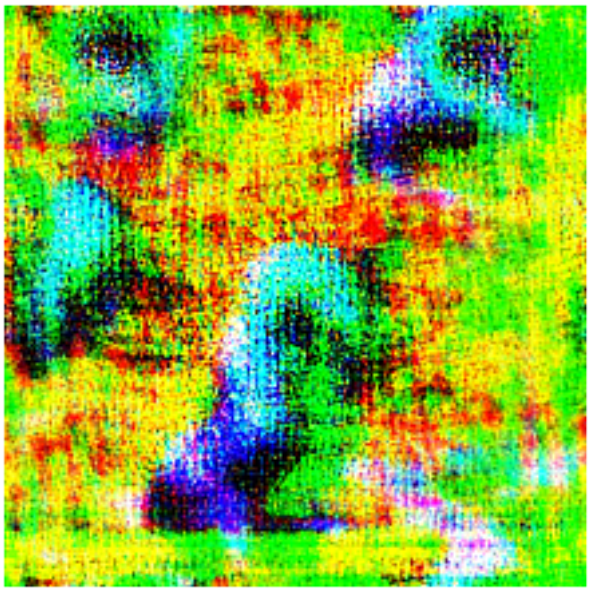
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



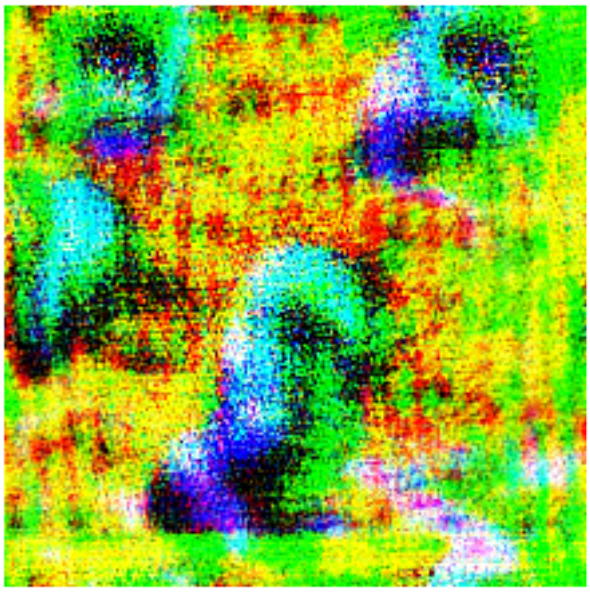
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 350 / 1000



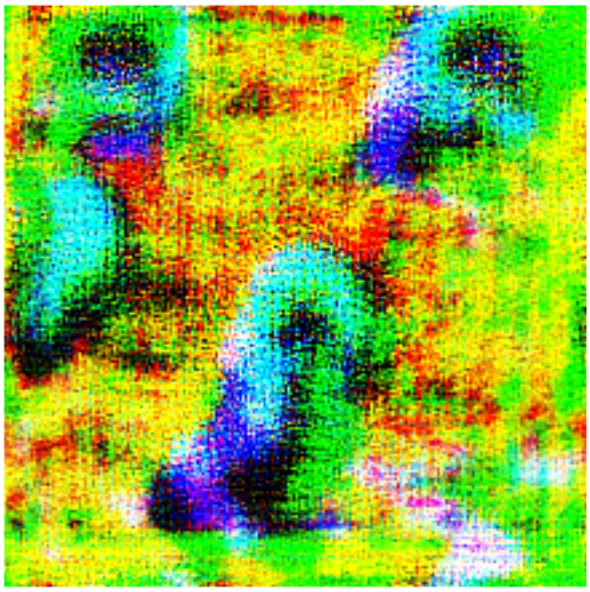
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 400 / 1000



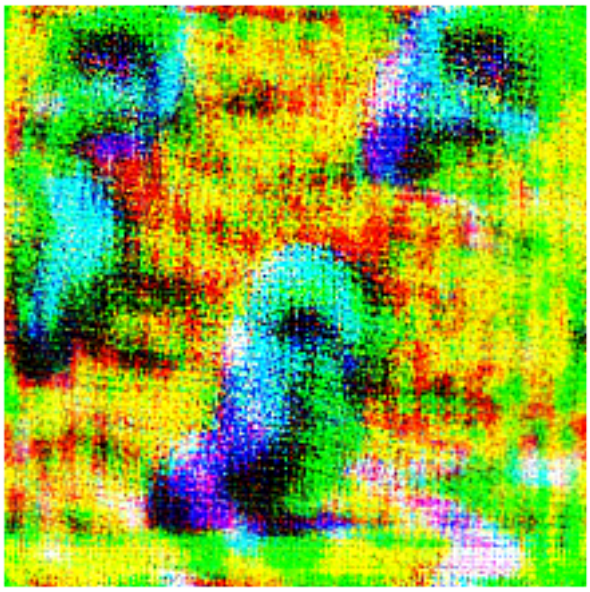
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 450 / 1000



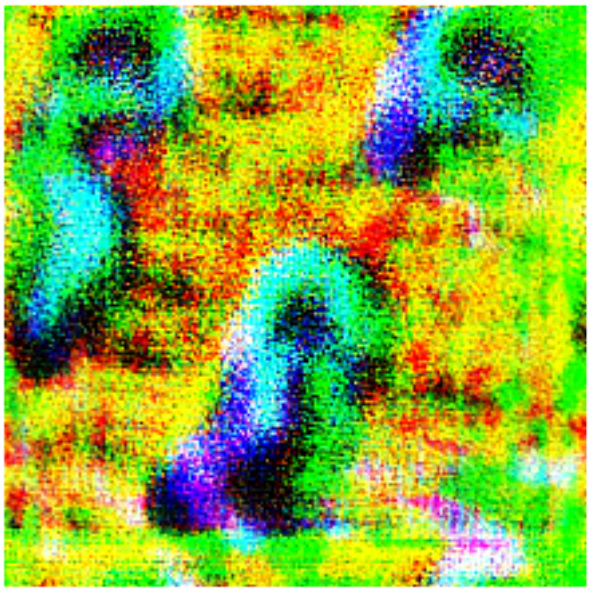
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



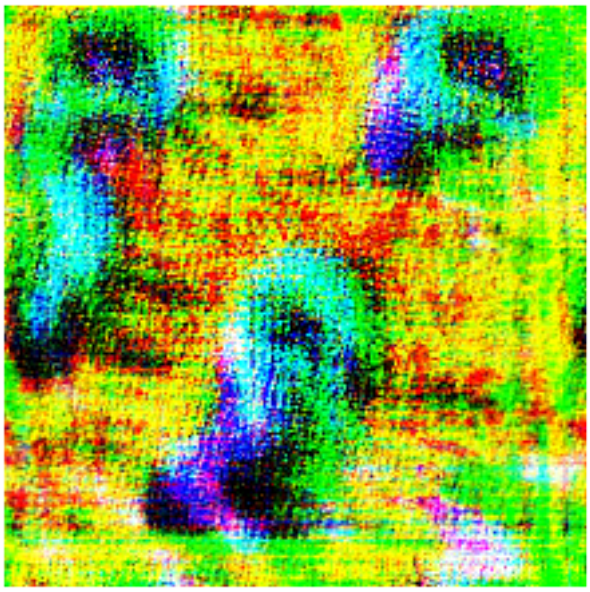
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



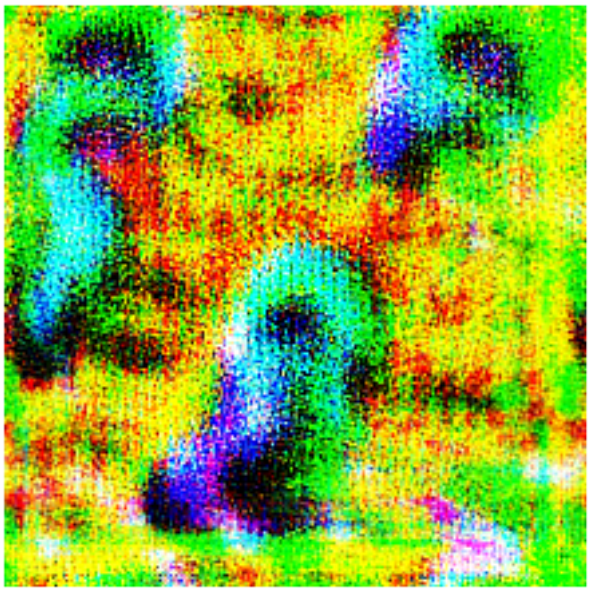
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



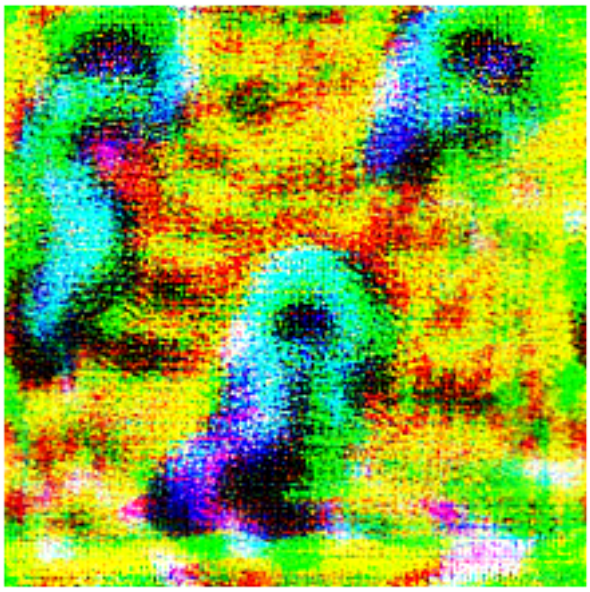
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



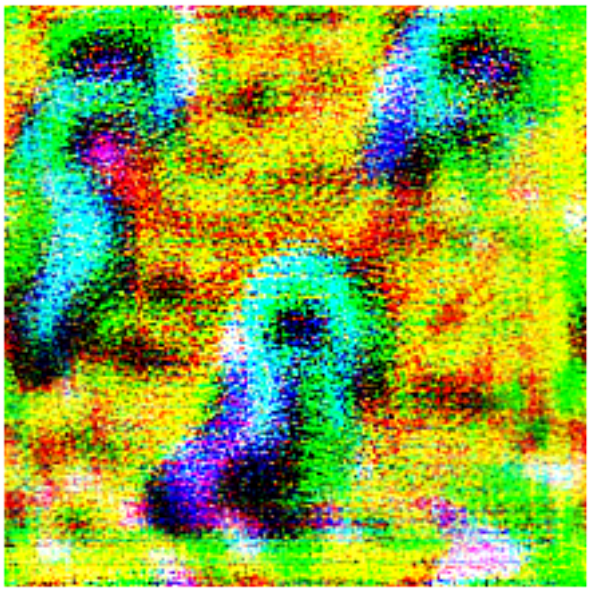
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



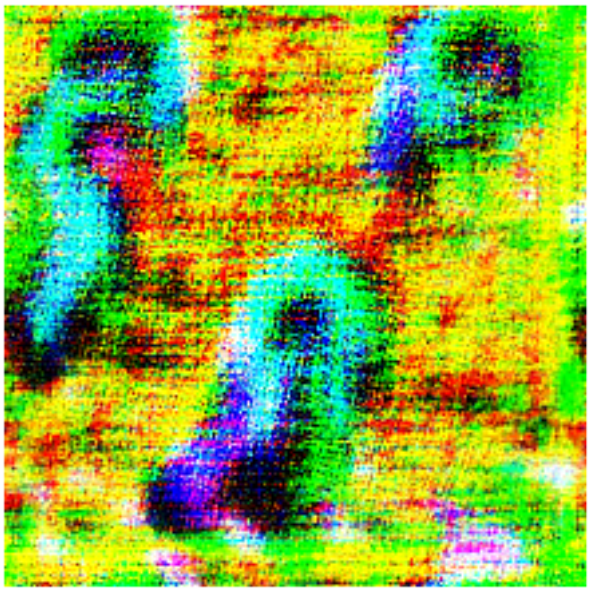
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 750 / 1000



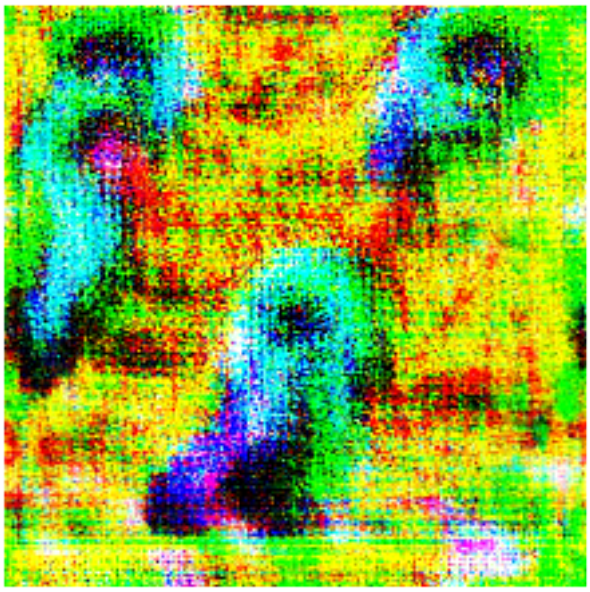
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 800 / 1000



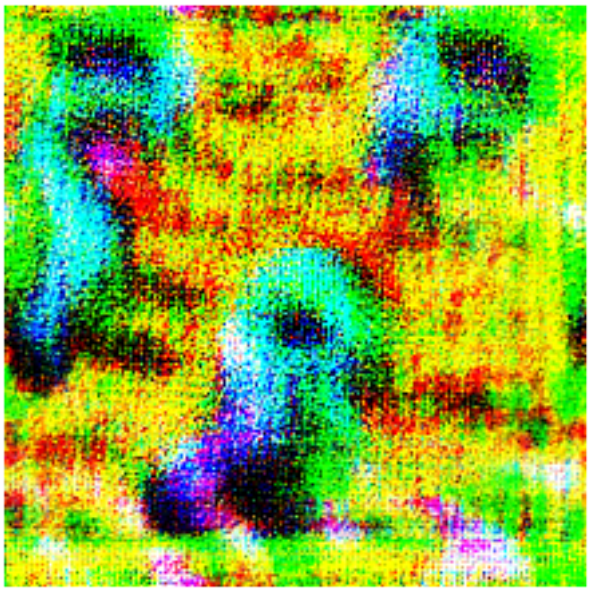
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 850 / 1000

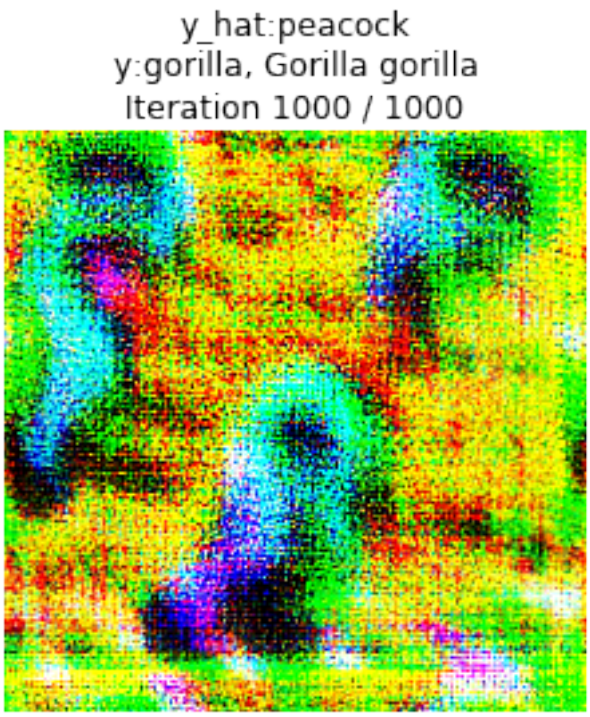


y_hat:jigsaw puzzle
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



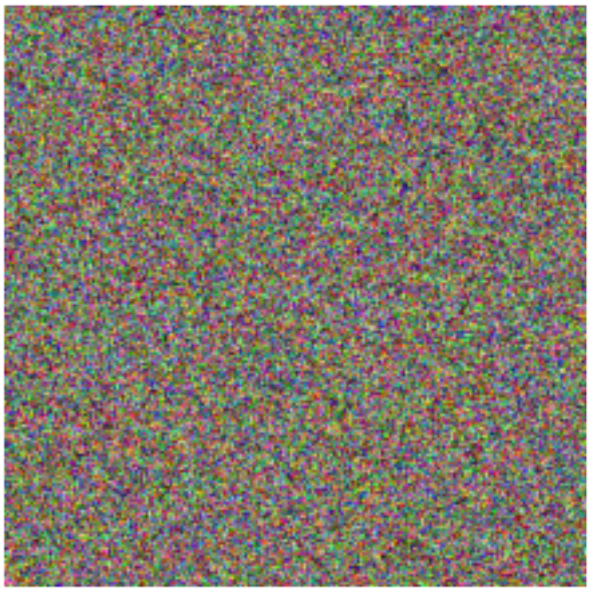
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 950 / 1000



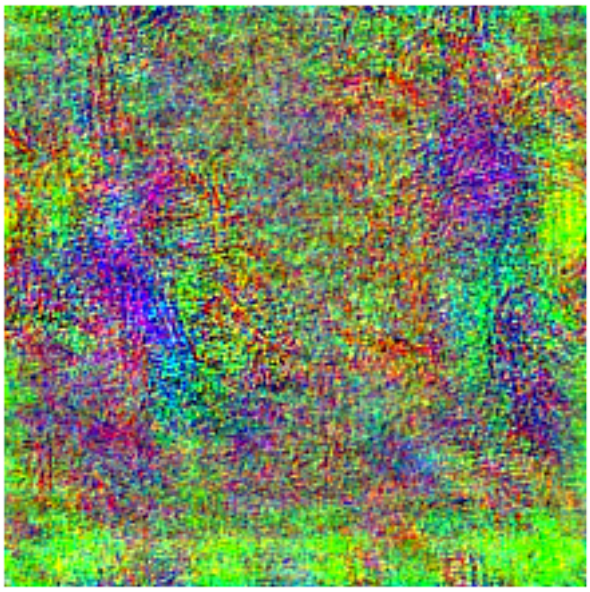


In [26]:
```
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to use GPU
model.type(dtype)

l2_reg = 1e-3
learning_rate = 5

num_iterations = 1000

#target_y = 281 # Tabby cat
# target_y = 187 # Yorkshire Terrier
#target_y = 76 # Tarantula
# target_y = 78 # Tick
# target_y = 683 # Oboe
target_y = 366 # Gorilla
# target_y = 604 # Hourglass
# target_y = np.random.randint(1000) # Classe aléatoire
out = create_class_visualization(target_y, model, dtype, show_every=50, num_iterations
```

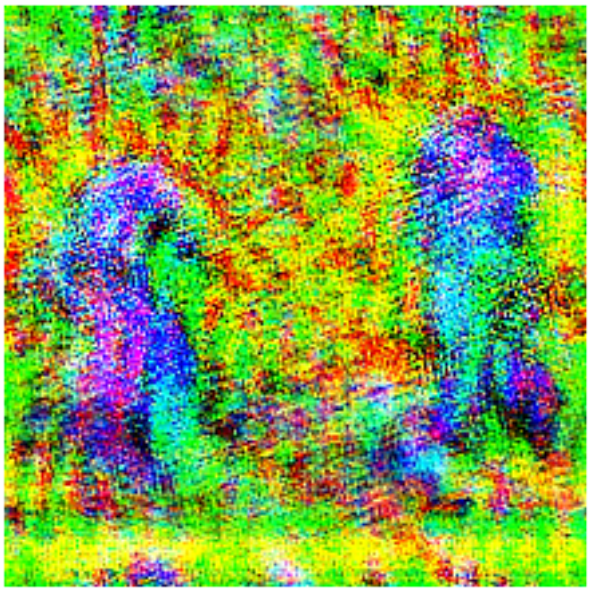
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 1 / 1000



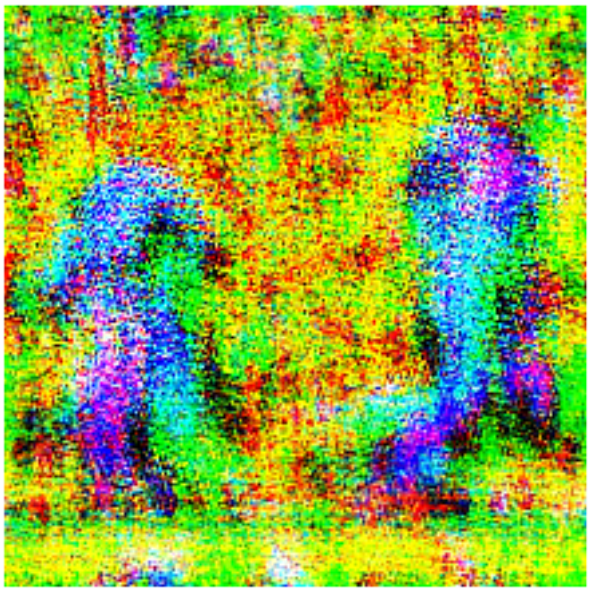
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



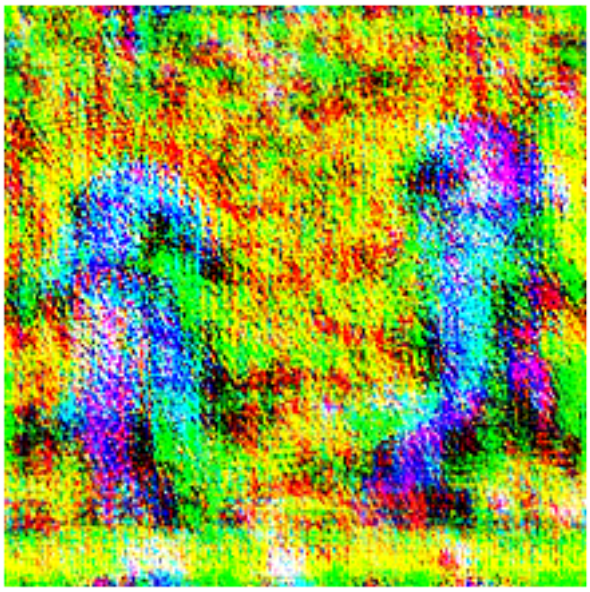
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



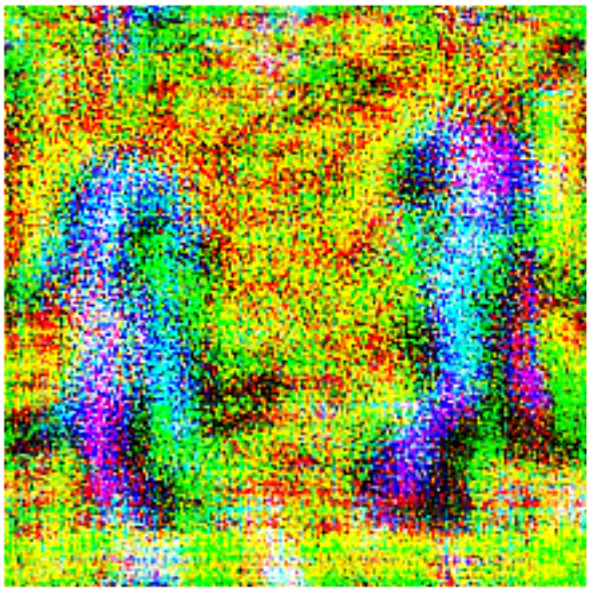
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 150 / 1000



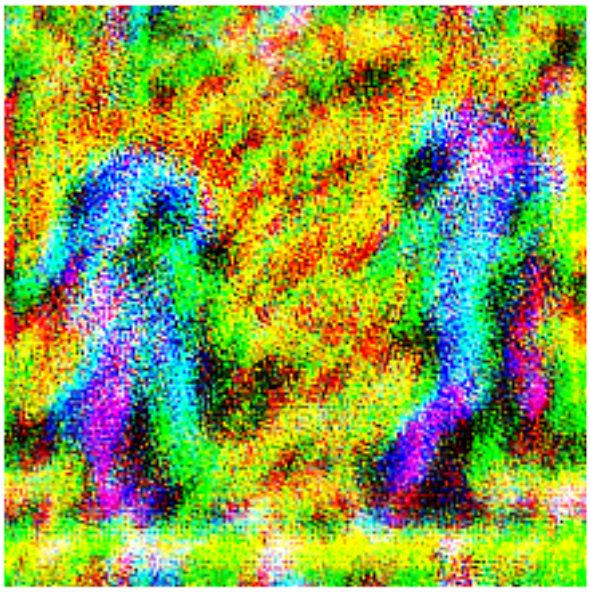
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 200 / 1000



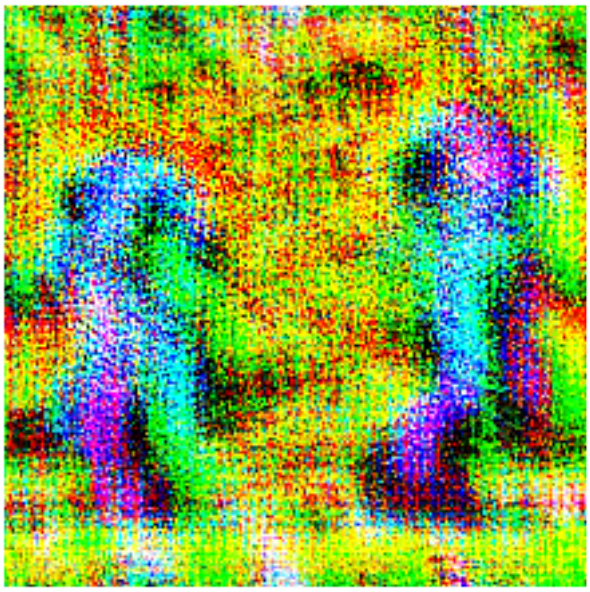
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



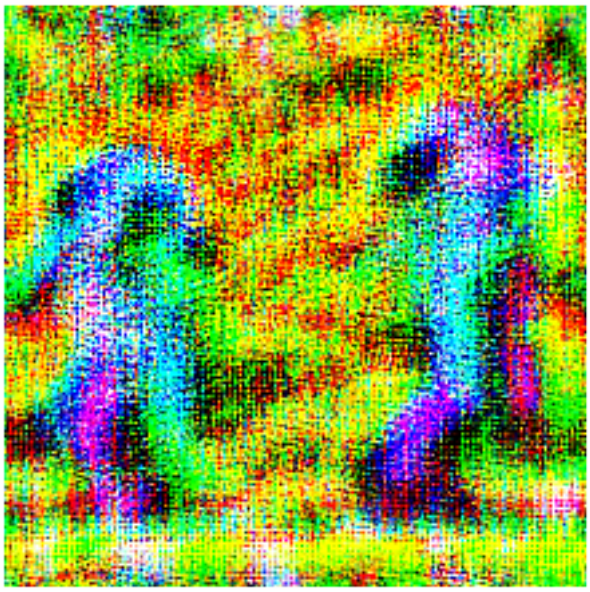
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



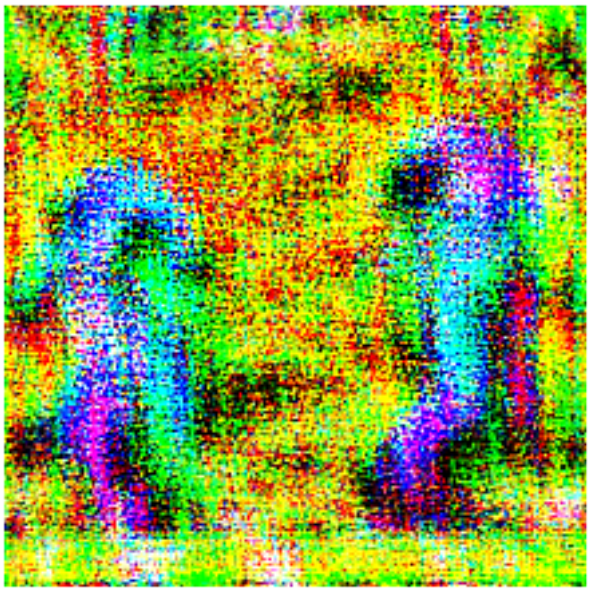
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 350 / 1000



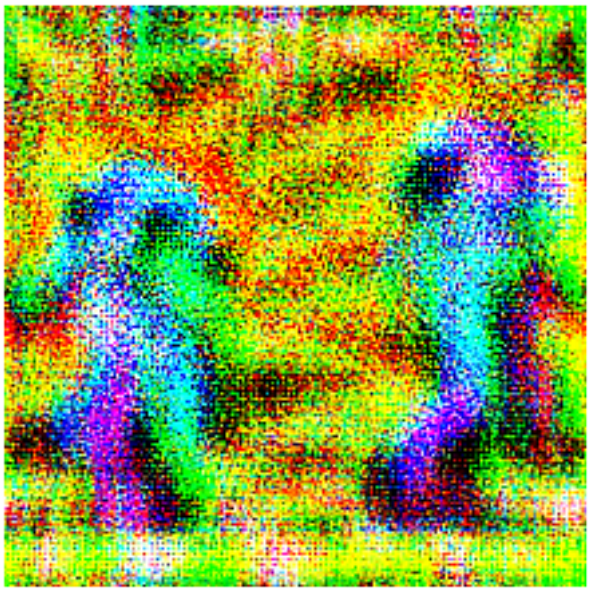
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 400 / 1000



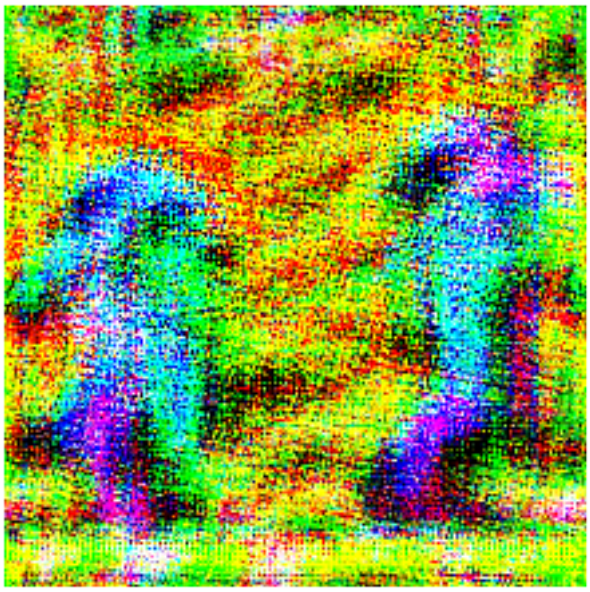
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 450 / 1000



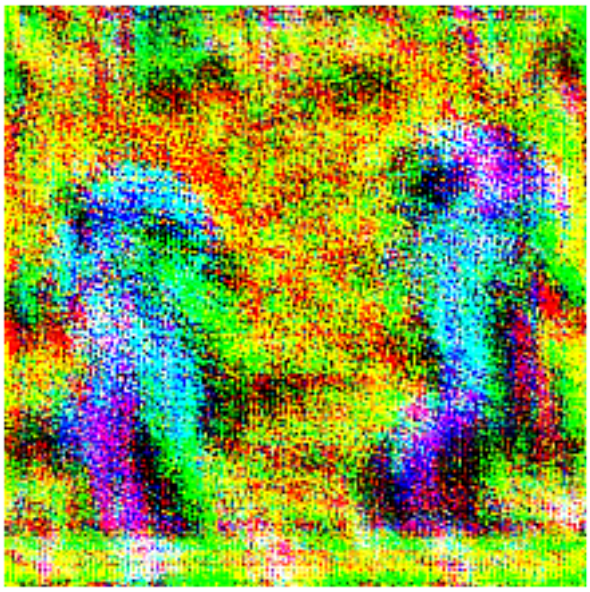
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



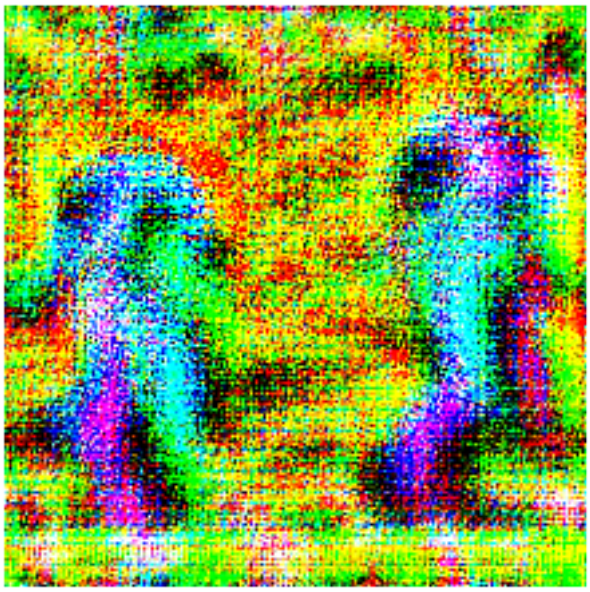
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



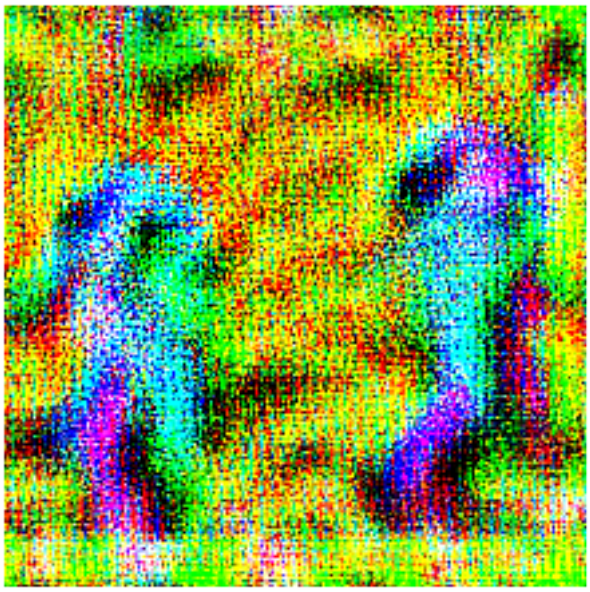
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



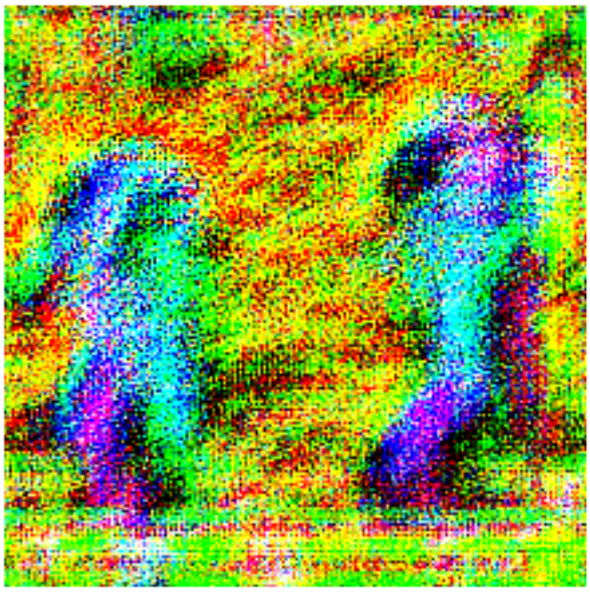
y_hat:standard poodle
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



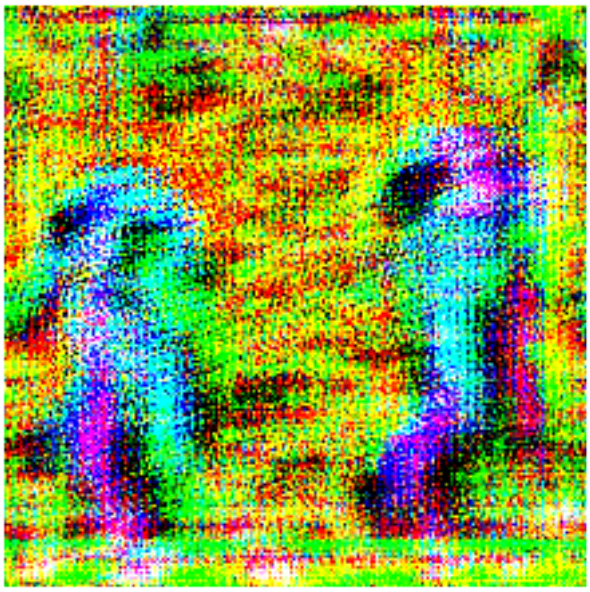
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



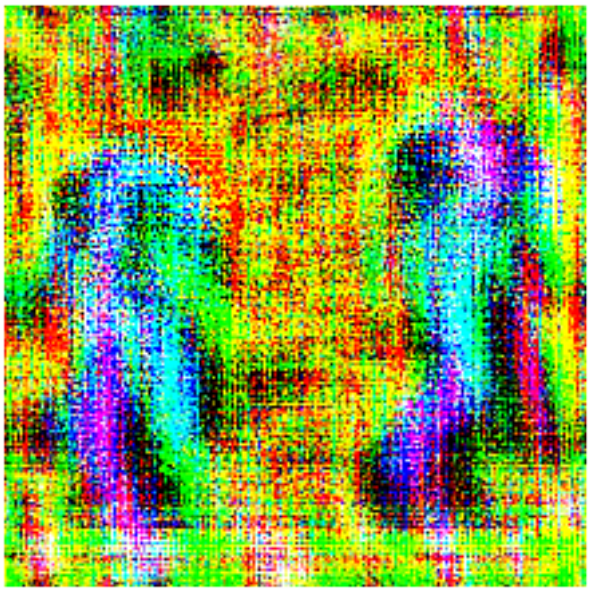
y_hat:jigsaw puzzle
y:gorilla, Gorilla gorilla
Iteration 750 / 1000



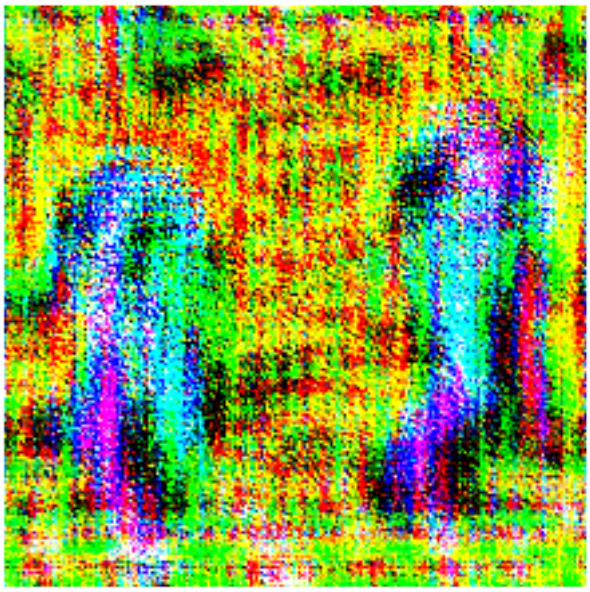
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 800 / 1000



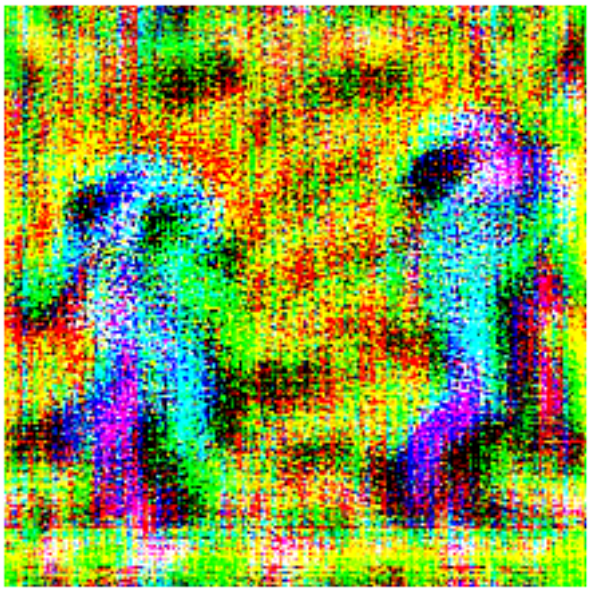
y_hat:peacock
y:gorilla, Gorilla gorilla
Iteration 850 / 1000

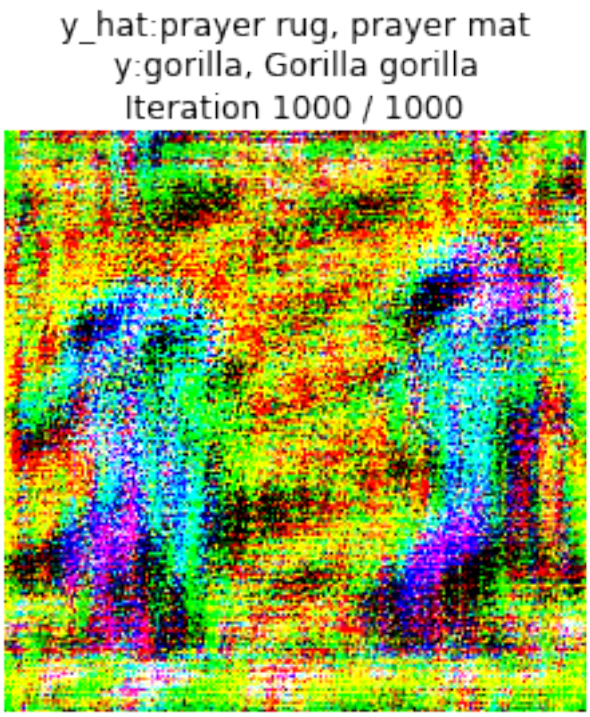


y_hat:standard poodle
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



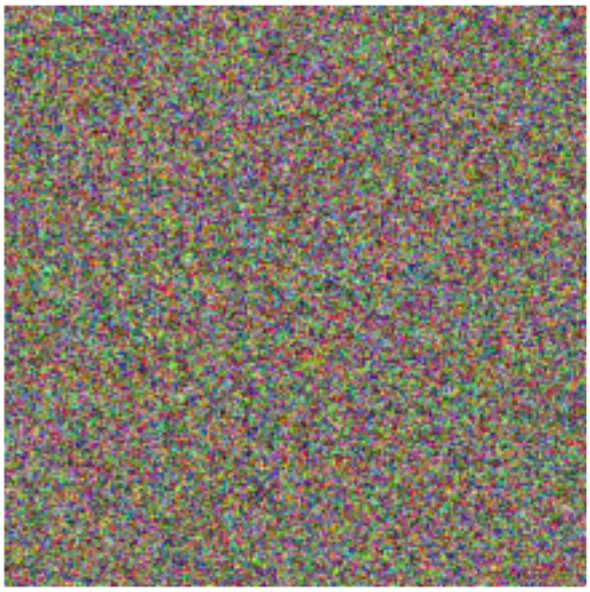
y_hat:prayer rug, prayer mat
y:gorilla, Gorilla gorilla
Iteration 950 / 1000



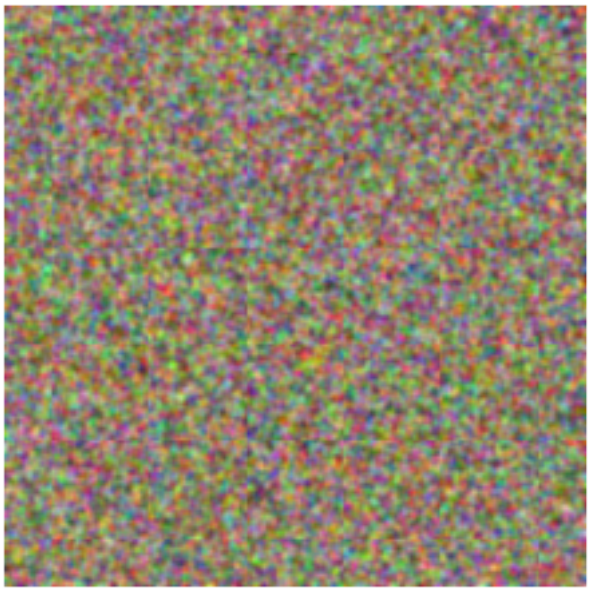


In [27]:
```
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to use GPU
model.type(dtype)

l2_reg = 1e-3
learning_rate = 0.01

num_iterations = 1000

#target_y = 281 # Tabby cat
# target_y = 187 # Yorkshire Terrier
#target_y = 76 # Tarantula
# target_y = 78 # Tick
# target_y = 683 # Oboe
target_y = 366 # Gorilla
# target_y = 604 # Hourglass
# target_y = np.random.randint(1000) # Classe aléatoire
out = create_class_visualization(target_y, model, dtype, show_every=50, num_iterations
```

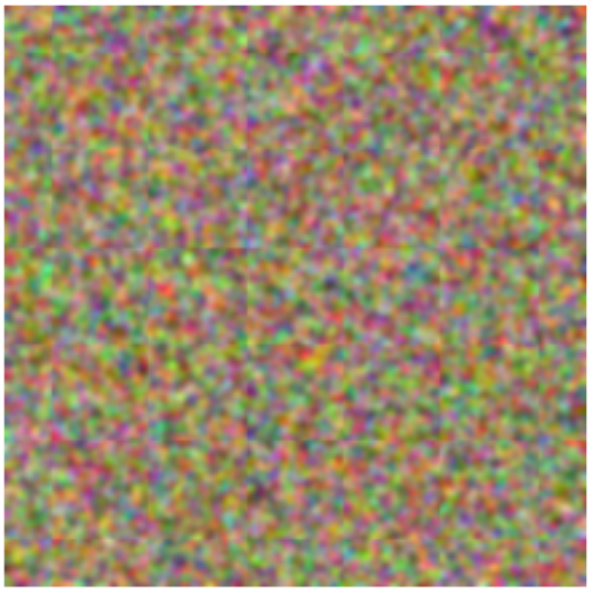
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 1 / 1000



y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



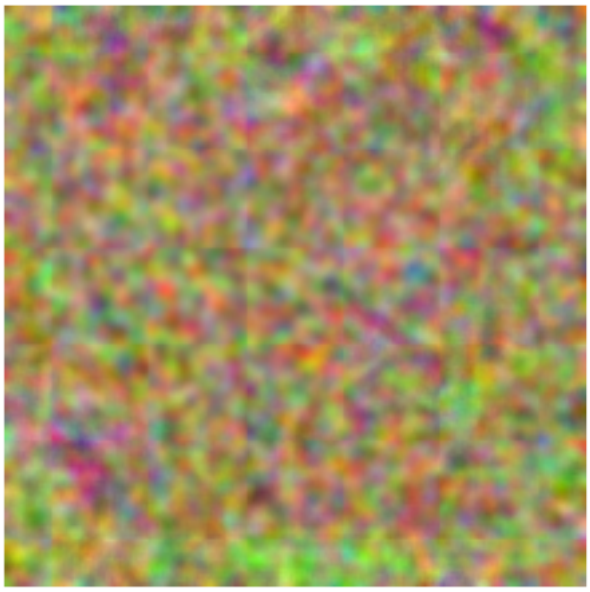
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



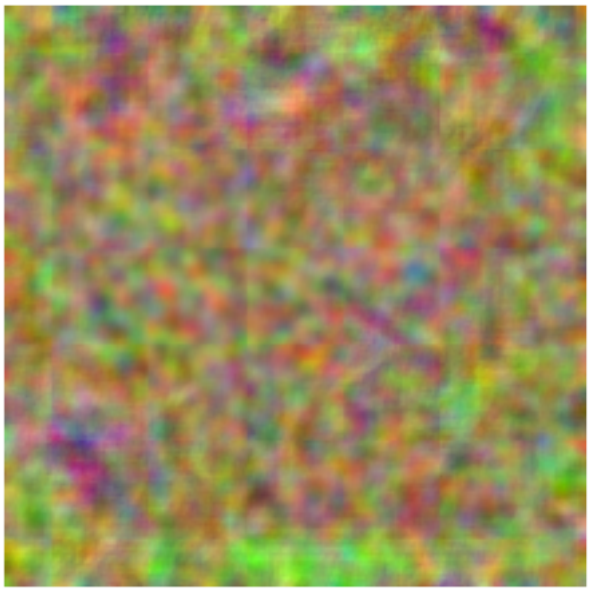
y_hat:croquet ball
y:gorilla, Gorilla gorilla
Iteration 150 / 1000

y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 200 / 1000

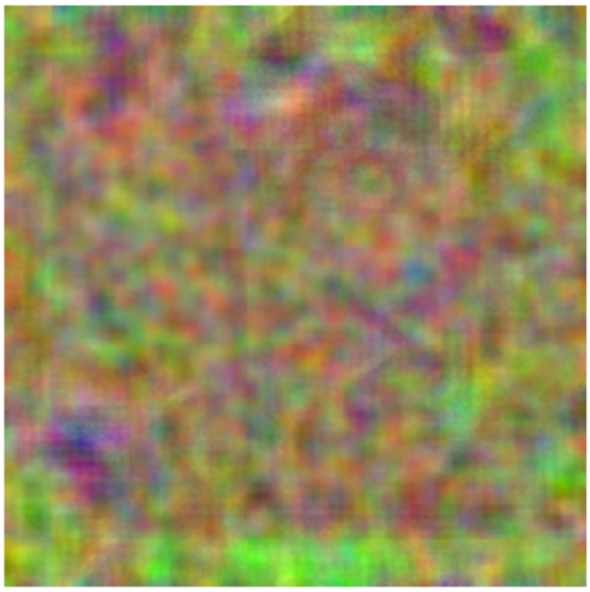
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



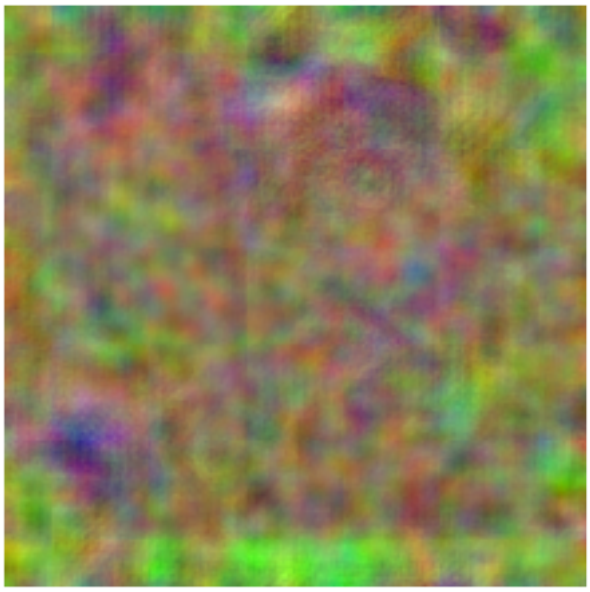
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



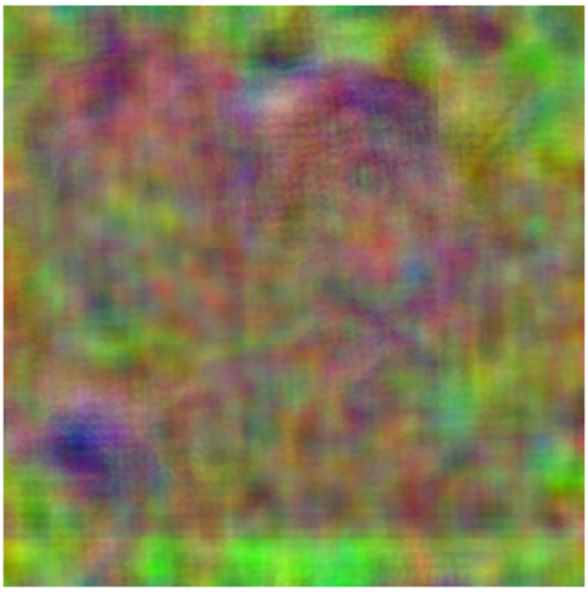
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 350 / 1000



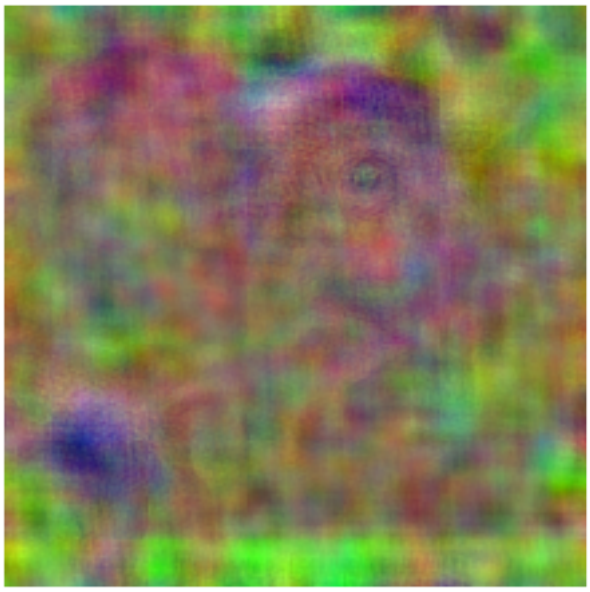
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 400 / 1000



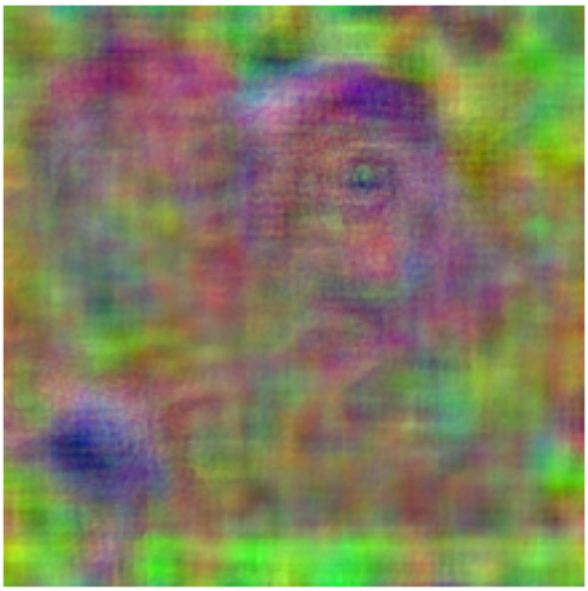
y_hat:patas, hussar monkey, Erythrocebus patas
y:gorilla, Gorilla gorilla
Iteration 450 / 1000



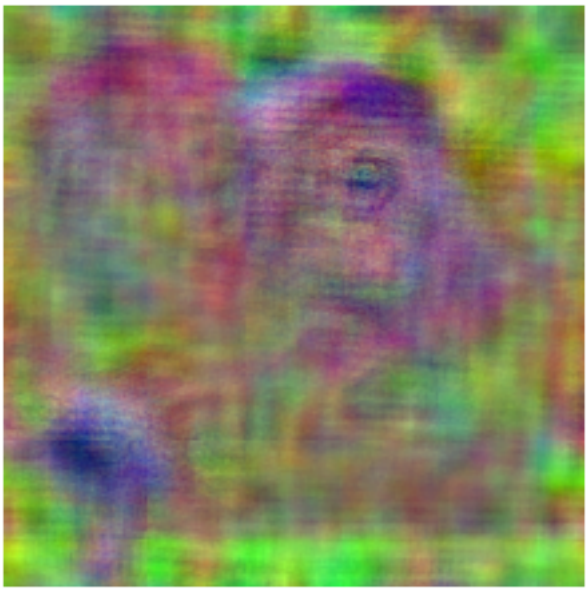
y_hat:baboon
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



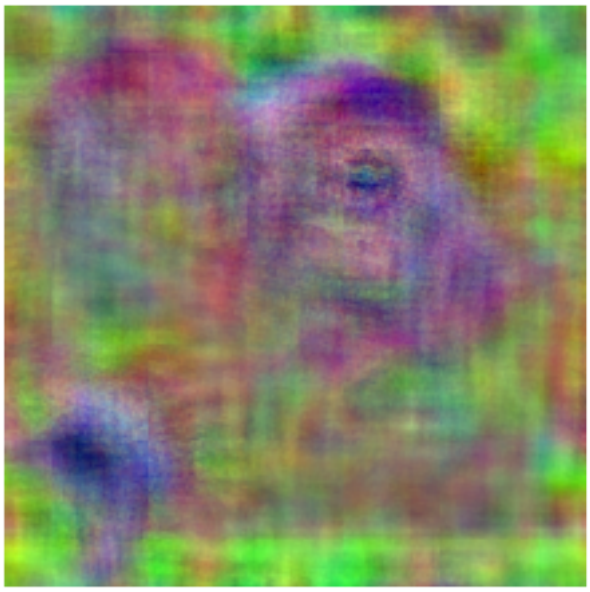
y_hat:patas, hussar monkey, Erythrocebus patas
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



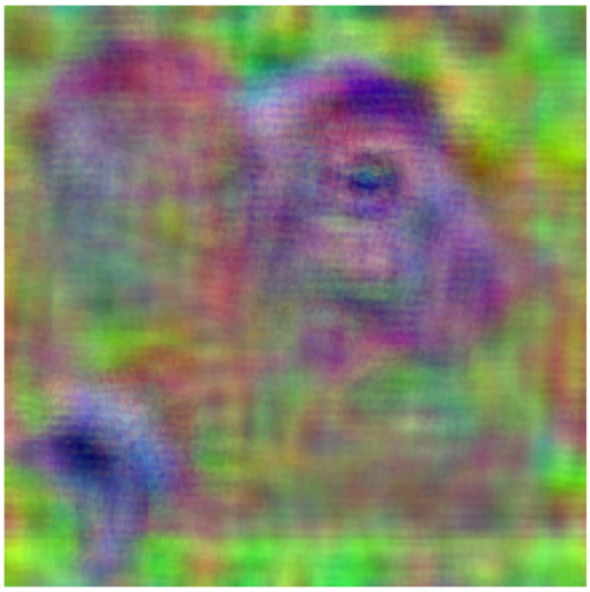
y_hat:patas, hussar monkey, Erythrocebus patas
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



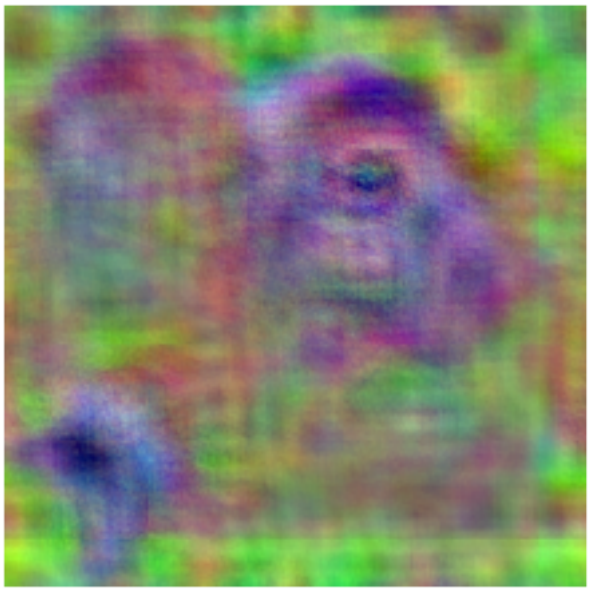
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



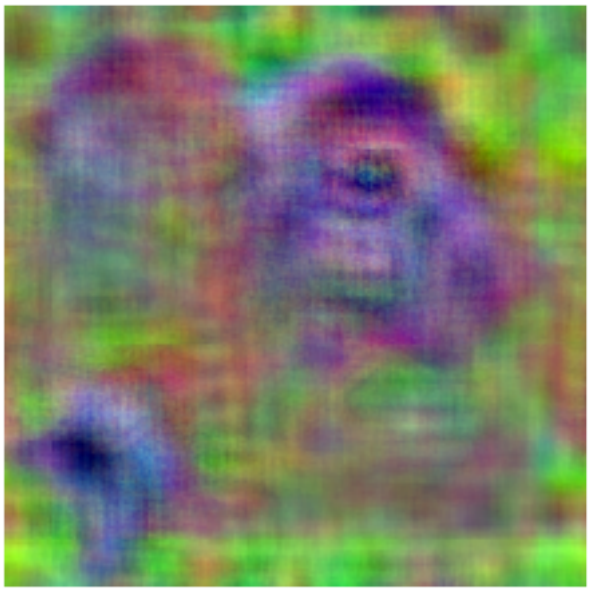
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



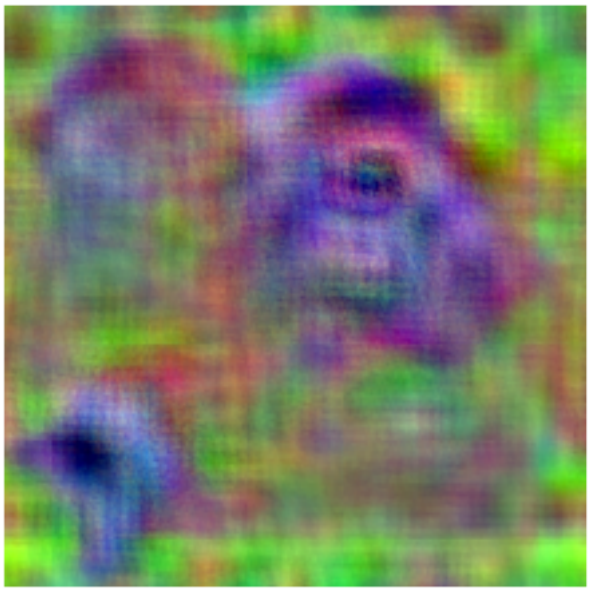
y_hat:macaque
y:gorilla, Gorilla gorilla
Iteration 750 / 1000



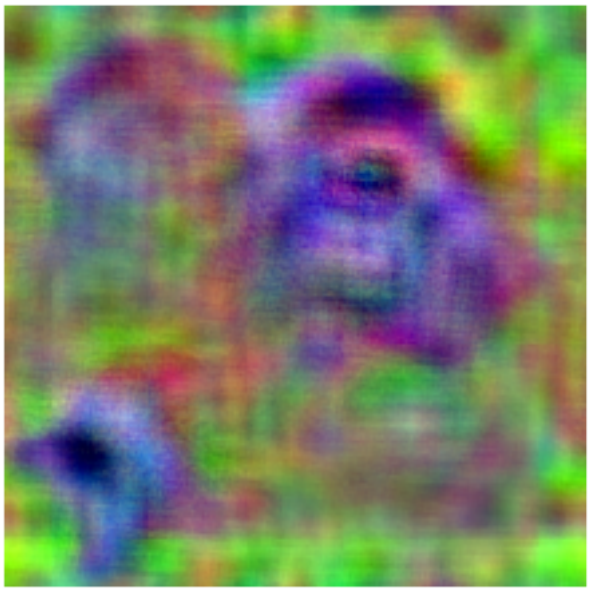
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 800 / 1000



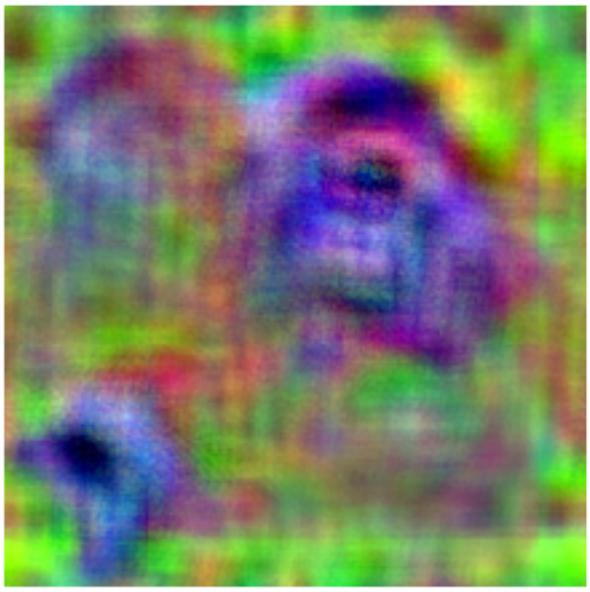
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 850 / 1000

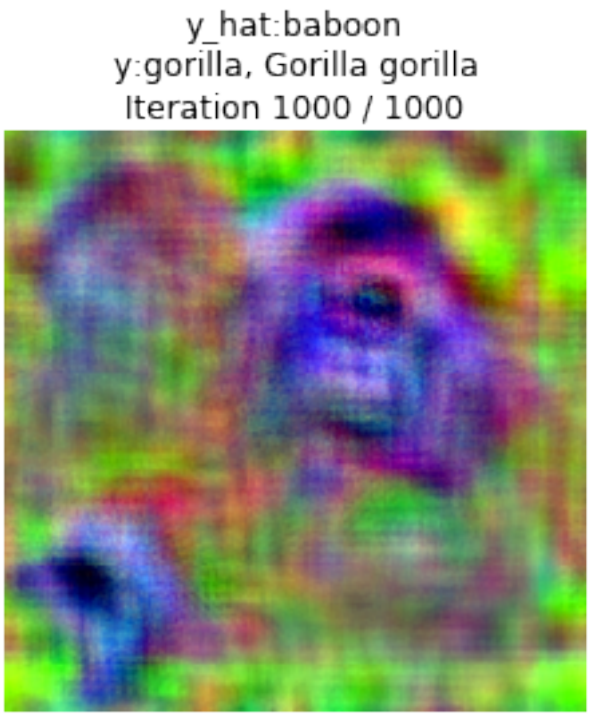


y_hat:macaque
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



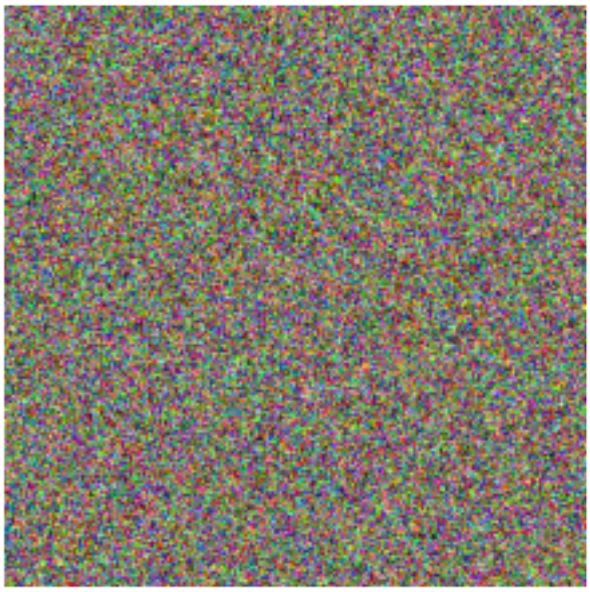
y_hat:gorilla, Gorilla gorilla
y:gorilla, Gorilla gorilla
Iteration 950 / 1000



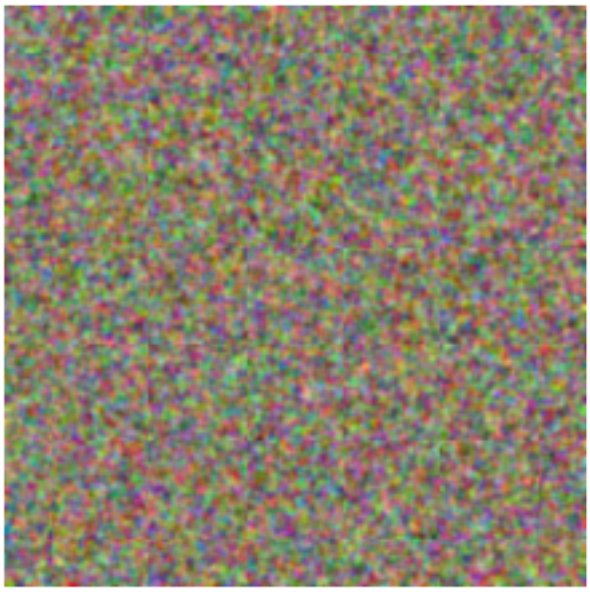


In [28]:
```
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to use GPU
model.type(dtype)

l2_reg = 1e-3
learning_rate = 0.001

num_iterations = 1000

#target_y = 281 # Tabby cat
# target_y = 187 # Yorkshire Terrier
#target_y = 76 # Tarantula
# target_y = 78 # Tick
# target_y = 683 # Oboe
target_y = 366 # Gorilla
# target_y = 604 # Hourglass
# target_y = np.random.randint(1000) # Classe aléatoire
out = create_class_visualization(target_y, model, dtype, show_every=50, num_iterations
```

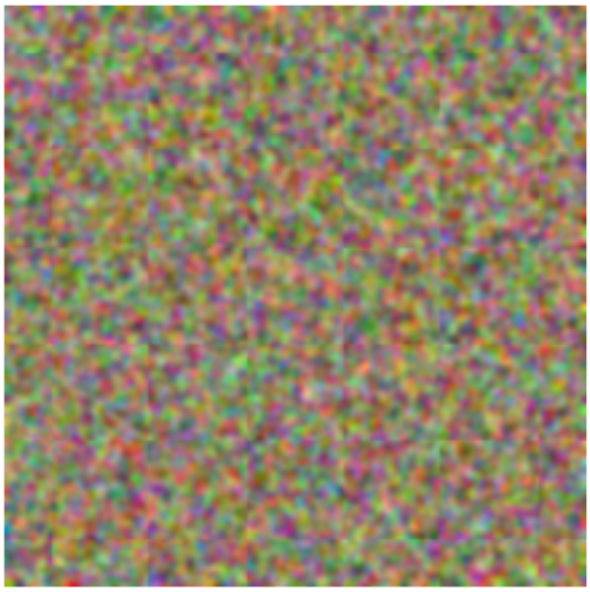
y_hat:poncho
y:gorilla, Gorilla gorilla
Iteration 1 / 1000



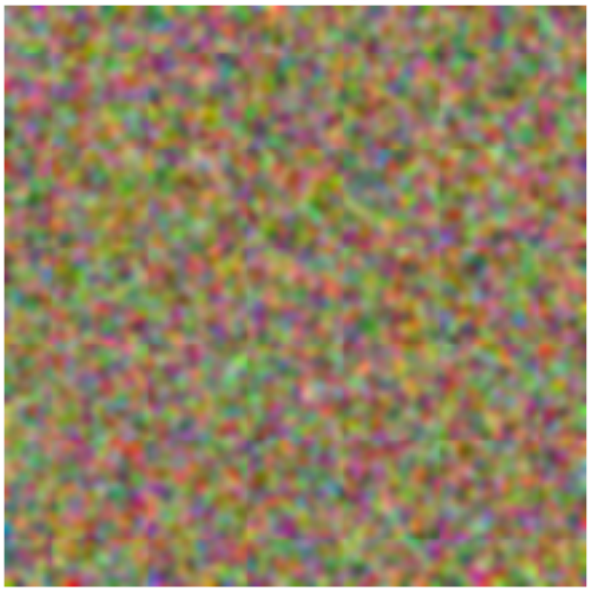
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 50 / 1000



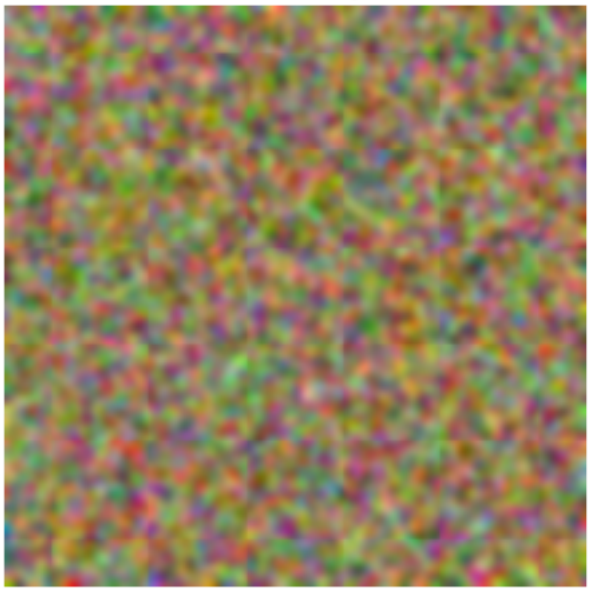
y_hat:doormat, welcome mat
y:gorilla, Gorilla gorilla
Iteration 100 / 1000



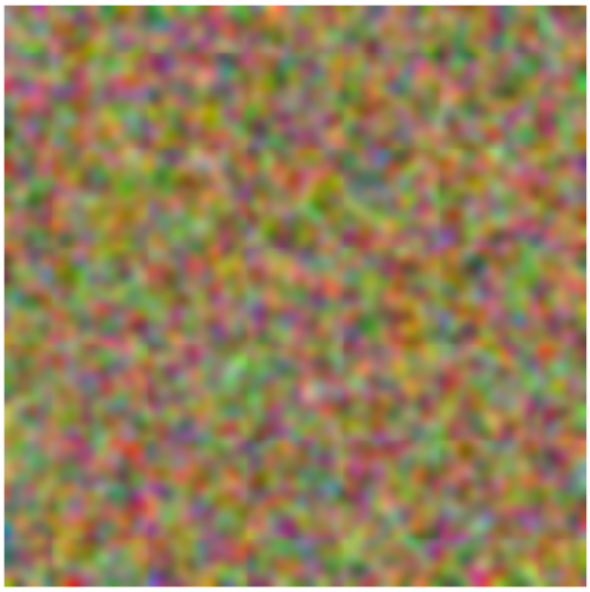
y_hat:leafhopper
y:gorilla, Gorilla gorilla
Iteration 150 / 1000



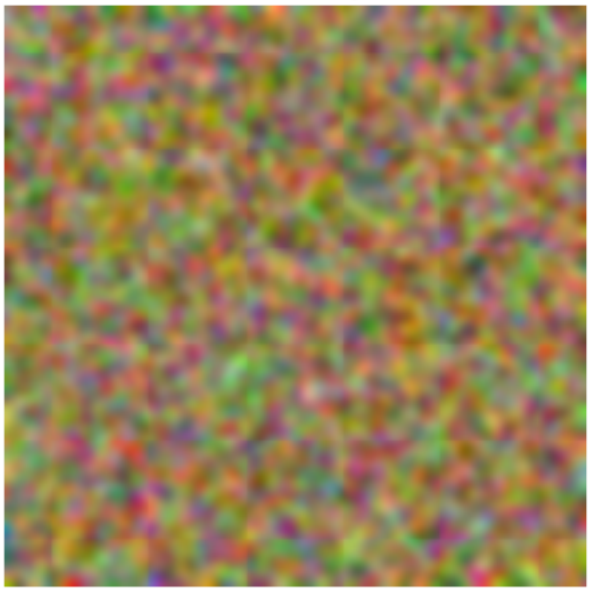
y_hat:golf ball
y:gorilla, Gorilla gorilla
Iteration 200 / 1000



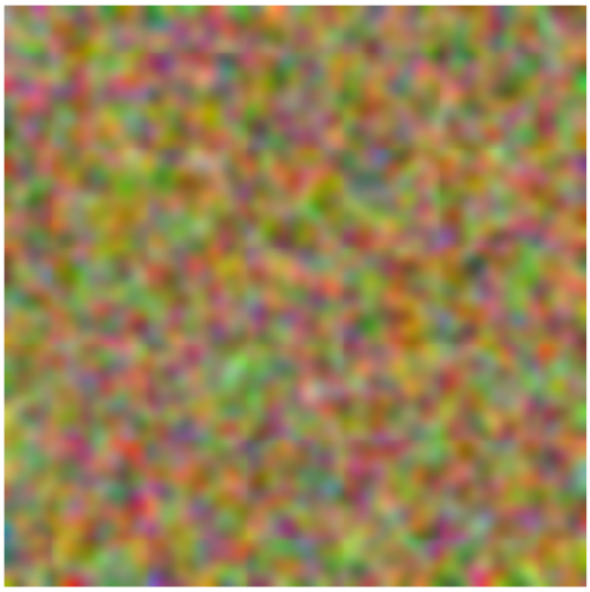
y_hat:papillon
y:gorilla, Gorilla gorilla
Iteration 250 / 1000



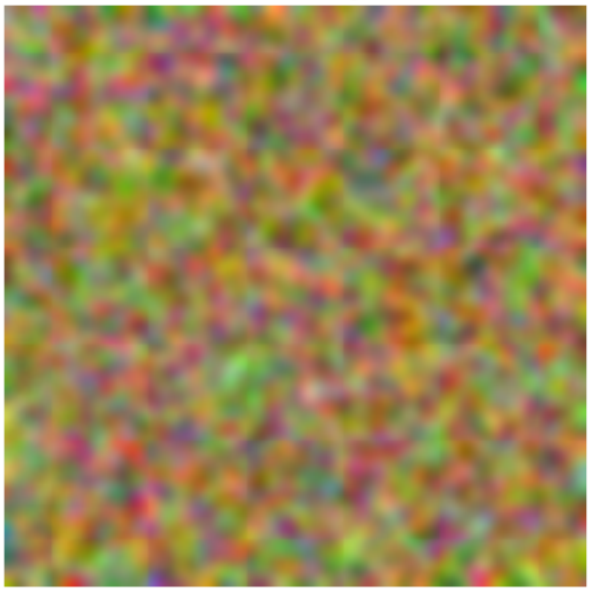
y_hat:papillon
y:gorilla, Gorilla gorilla
Iteration 300 / 1000



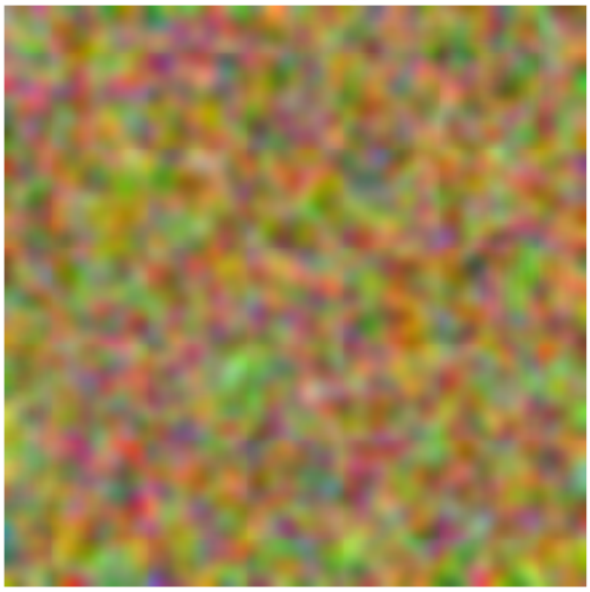
y_hat:racket, racquet
y:gorilla, Gorilla gorilla
Iteration 350 / 1000



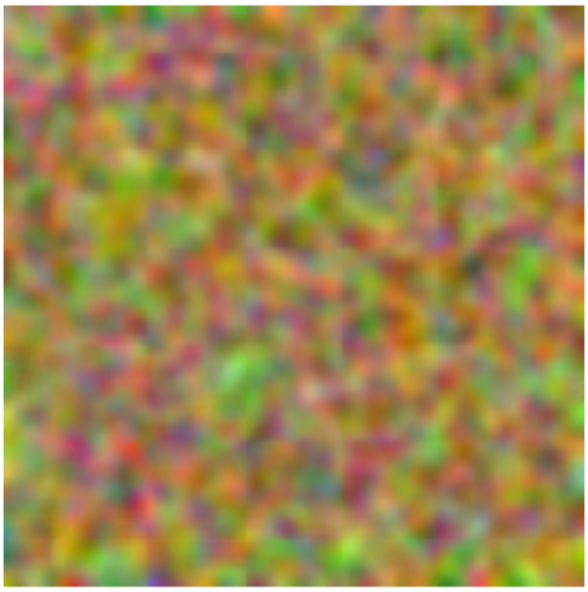
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 400 / 1000



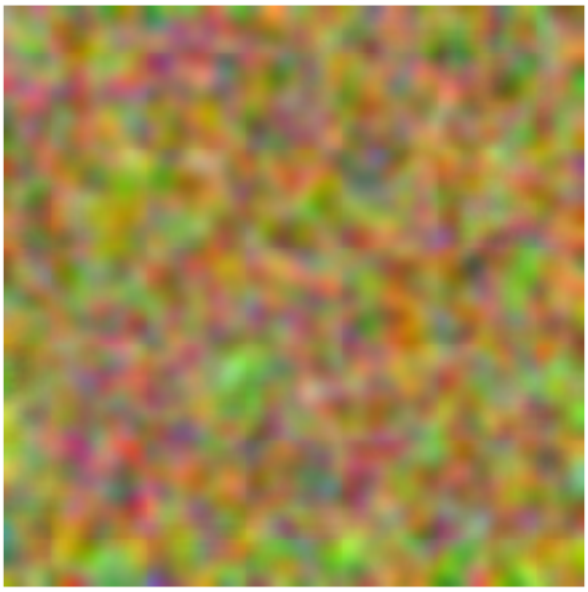
y_hat:hare
y:gorilla, Gorilla gorilla
Iteration 450 / 1000



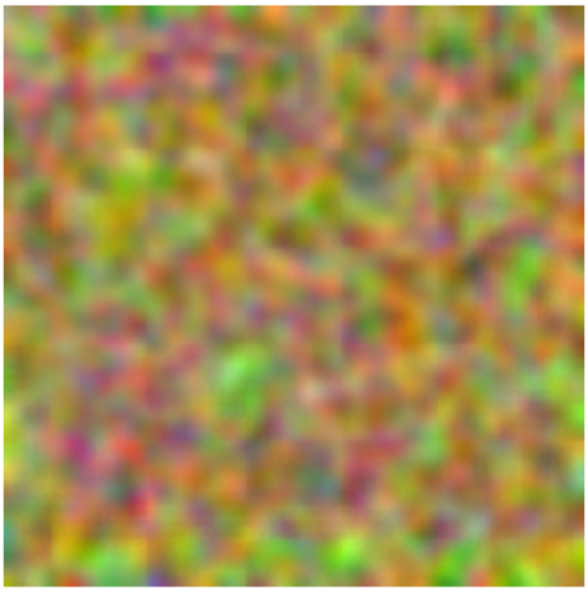
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 500 / 1000



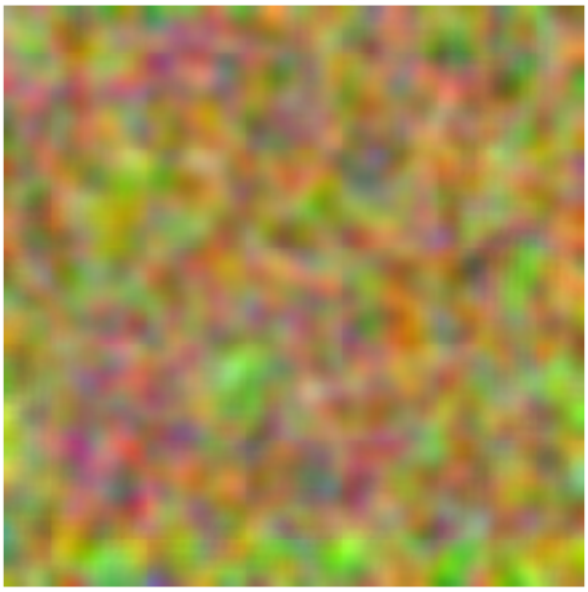
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 550 / 1000



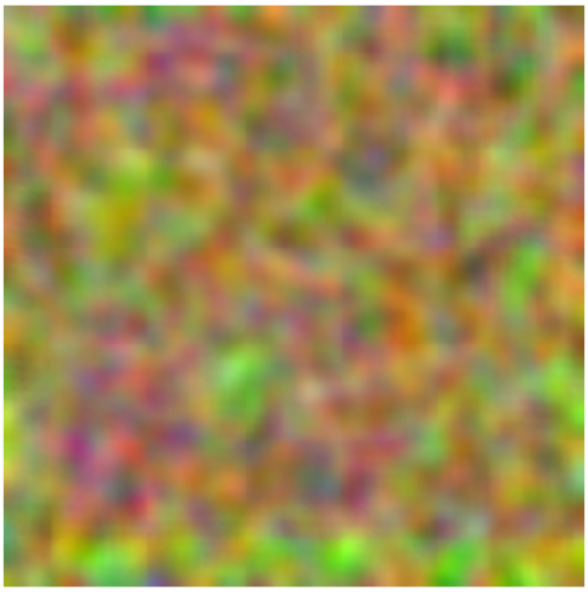
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 600 / 1000



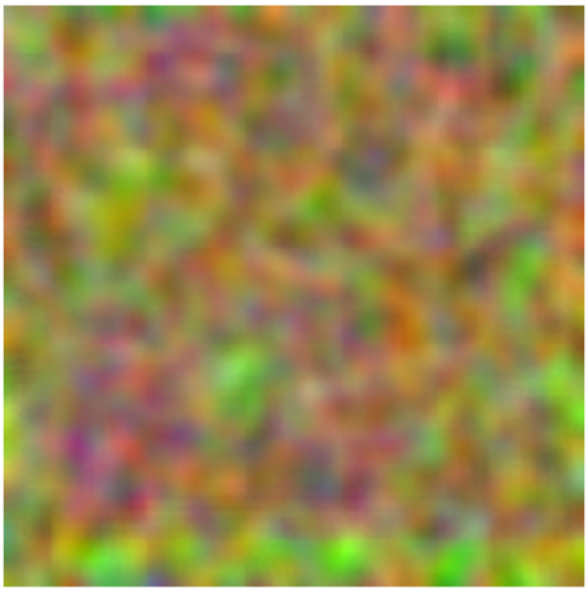
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 650 / 1000



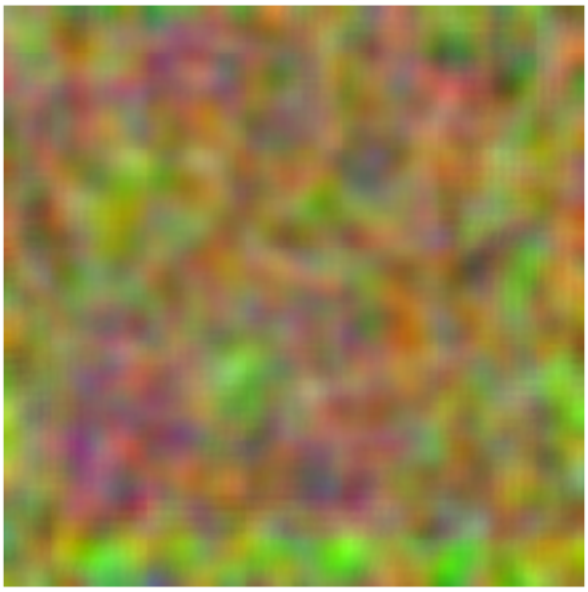
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 700 / 1000



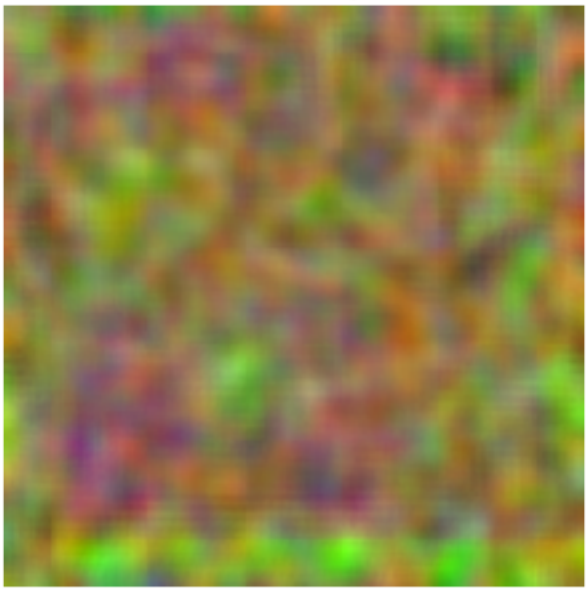
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 750 / 1000



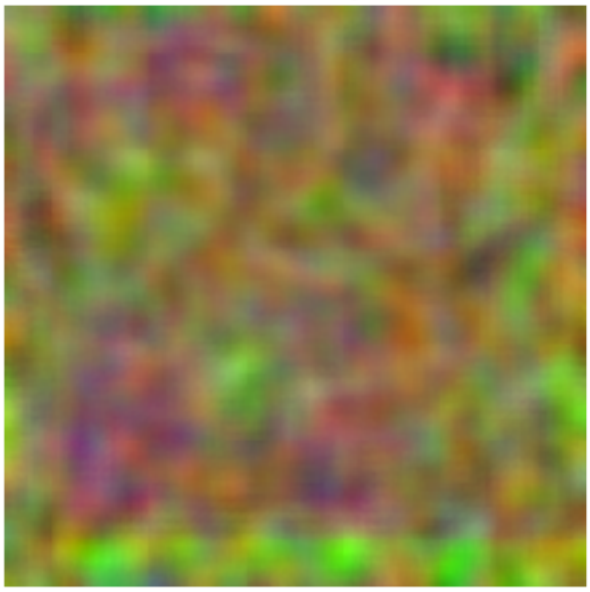
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 800 / 1000



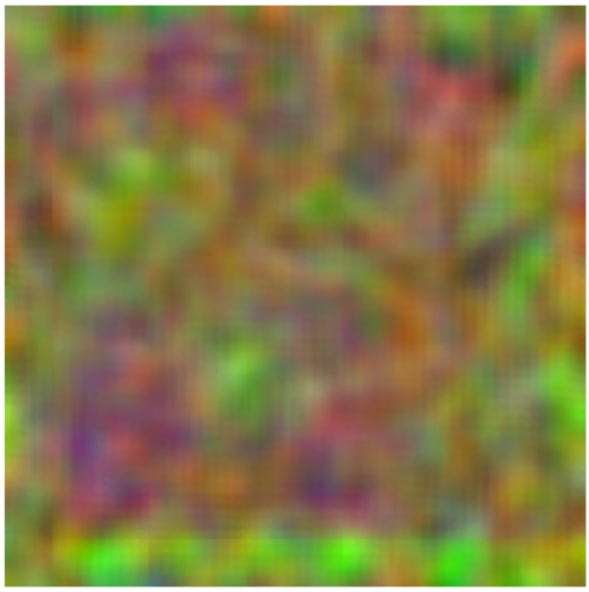
y_hat:prairie chicken, prairie grouse, prairie fowl
y:gorilla, Gorilla gorilla
Iteration 850 / 1000



y_hat:dhole, Cuon alpinus
y:gorilla, Gorilla gorilla
Iteration 900 / 1000



y_hat:fox squirrel, eastern fox squirrel, Sciurus niger
y:gorilla, Gorilla gorilla
Iteration 950 / 1000



y_hat:fox squirrel, eastern fox squirrel, Sciurus niger
y:gorilla, Gorilla gorilla
Iteration 1000 / 1000