ta-project

February 17, 2024

[ ]:
```
import sys
print(sys.version)
```

```
3.7.16 (default, Jan 17 2023, 09:28:58)
[Clang 14.0.6 ]
```

[ ]:
```
pip install scratchai-nightly
```

```
Requirement already satisfied: scratchai-nightly in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (0.0.1a3)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install torchvision==0.9.1
```

```
Requirement already satisfied: torchvision==0.9.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (0.9.1)
Requirement already satisfied: pillow>=4.1.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
torchvision==0.9.1) (9.5.0)
Requirement already satisfied: numpy in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
torchvision==0.9.1) (1.21.6)
Requirement already satisfied: torch==1.8.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
torchvision==0.9.1) (1.8.1)
Requirement already satisfied: typing-extensions in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
torch==1.8.1->torchvision==0.9.1) (4.7.1)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install flashtorch
```

```
Requirement already satisfied: flashtorch in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (0.1.3)
Requirement already satisfied: matplotlib in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (3.5.3)
Requirement already satisfied: Pillow in
```

```
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (9.5.0)
Requirement already satisfied: importlib-resources in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (5.12.0)
Requirement already satisfied: torchvision in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (0.9.1)
Requirement already satisfied: torch in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (1.8.1)
Requirement already satisfied: numpy in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
flashtorch) (1.21.6)
Requirement already satisfied: zipp>=3.1.0 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
importlib-resources->flashtorch) (3.15.0)
Requirement already satisfied: python-dateutil>=2.7 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (1.4.5)
Requirement already satisfied: pyparsing>=2.2.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (3.1.1)
Requirement already satisfied: cycler>=0.10 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (4.38.0)
Requirement already satisfied: packaging>=20.0 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
matplotlib->flashtorch) (23.2)
Requirement already satisfied: typing-extensions in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
torch->flashtorch) (4.7.1)
Requirement already satisfied: six>=1.5 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
python-dateutil>=2.7->matplotlib->flashtorch) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install tqdm
```

```
Requirement already satisfied: tqdm in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (4.66.2)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install mapextrackt
```

```
Requirement already satisfied: mapextrackt in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (0.4.8.6)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install requests
```

```
Requirement already satisfied: requests in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (2.31.0)
Requirement already satisfied: idna<4,>=2.5 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
requests) (3.6)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
requests) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
requests) (2024.2.2)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages
(4.9.0.80)
Requirement already satisfied: numpy>=1.17.0 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
opencv-python) (1.21.6)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
pip install scipy
```

```
Requirement already satisfied: scipy in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (1.7.3)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-packages (from
scipy) (1.21.6)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
import os
import requests

stop_sign_url = 'https://static01.nyt.com/images/2011/12/11/magazine/11wmt1/
↪mag-11WMT-t_CA0-jumbo.jpg'
```

```
output_folder = 'input_images'

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

# Download the image using requests
response = requests.get(stop_sign_url)

if response.status_code == 200:
    # Save the image to the output folder
    output_path = os.path.join(output_folder, 'stop.jpg')
    with open(output_path, 'wb') as f:
        f.write(response.content)
    print('Download successful!')
else:
    print(f'Failed to download image. Status code: {response.status_code}')
# suppress error
import logging as logging
import sys as sys
logging.disable(sys.maxsize)
```

```
Download successful!
```

[ ]:
```
# suppress error
import logging as logging
import sys as sys
logging.disable(sys.maxsize)

# import the library
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models
from scratchai import *

from flashtorch.activmax import GradientAscent
from MapExtrackt import FeatureExtractor
from torch.distributions import Normal


# set parameters
stop_sign_path = 'input_images/stop.jpg' #stop sign image path
true_class = 919 # imagenet id for street sign


# function handle to get prediction more easily
def get_prediction(image, model):
```

```
    #assumes img and net are datasets and models trained using imagenet dataset
    confidences = model(image.unsqueeze(0))
    class_idx = torch.argmax(confidences, dim=1).item()
    class_label = datasets.labels.imagenet_labels[class_idx]
    return class_label, confidences[0, class_idx].item(), class_idx
```

```
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[ ]:

1a: Making prediction

[ ]:
```
# load and preprocess the stop sign image
img = imgutils.load_img(stop_sign_path)
img = imgutils.get_trf('rz256_cc224_tt_normimgnet')(img) #normalize and reshape␣
 ↪the input image
```

[ ]:
```
from torchvision import models
resnet = models.resnet18(pretrained=True).eval()
```

[ ]:
```
get_prediction(img, resnet)
```

[ ]:
```
('street sign', 13.558082580566406, 919)
```

1b: Random perturbation

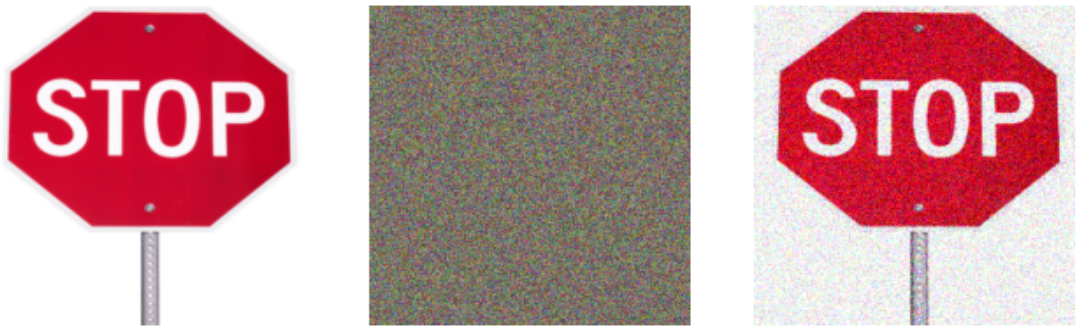
[ ]:
```
epsilon = 1
torch.manual_seed(0)
noisy_img = attacks.noise(img, eps=epsilon)
```

[ ]:
```
get_prediction(noisy_img, resnet)
```

[ ]:
```
('doormat, welcome mat', 16.598552703857422, 539)
```

[ ]:
```
from scratchai import *
```

[ ]:
```
imgutils.imshow([img, noisy_img-img, noisy_img], normd=True)
```

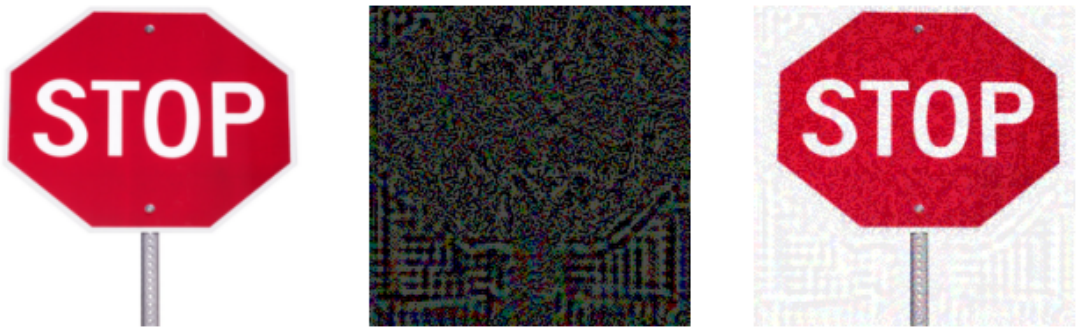


[ ]:

1c: FGM Attack

[ ]:
```
image_path =  stop_sign_path

images, true_labels, predicted_labels = one_call.attack(image_path, atk =␣
↪attacks.FGM, nstr = 'resnet18', ret=True)
```

[ ]:
```
imgutils.imshow(images)
```



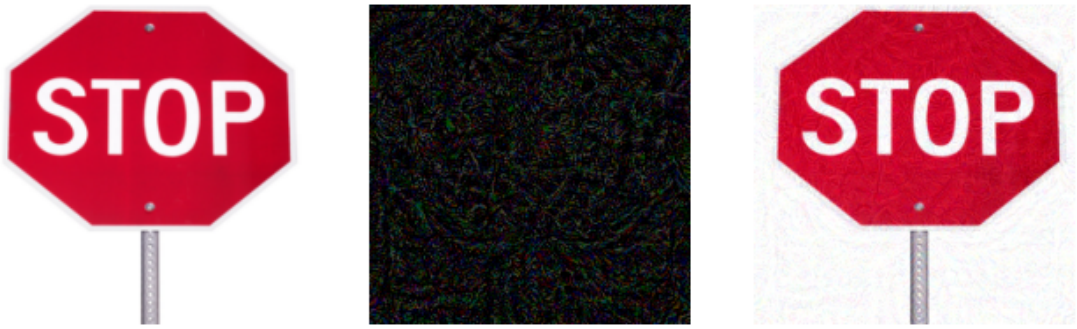
[ ]:
```
true_labels, predicted_labels
```

[ ]:
```
(('street sign', 13.558082580566406),
 ('doormat, welcome mat', 14.28164291381836))
```

1d: PGD Attack

[ ]:
```
target_class = 829

images, true_labels, predicted_labels = one_call.attack(image_path, atk =␣
↪attacks.PGD, nstr = 'resnet18', ret=True)
```

[ ]:
```
imgutils.imshow(images)
```



[ ]:
```
true_labels, predicted_labels
```

[ ]:
```
(('street sign', 13.558082580566406),
 ('jersey, T-shirt, tee shirt', 46.43096923828125))
```

2a: AlexNet layer 0 visualization

[ ]:
```
alexnet = models.alexnet(pretrained=True).eval()
```

[ ]:
```
print(alexnet)
```

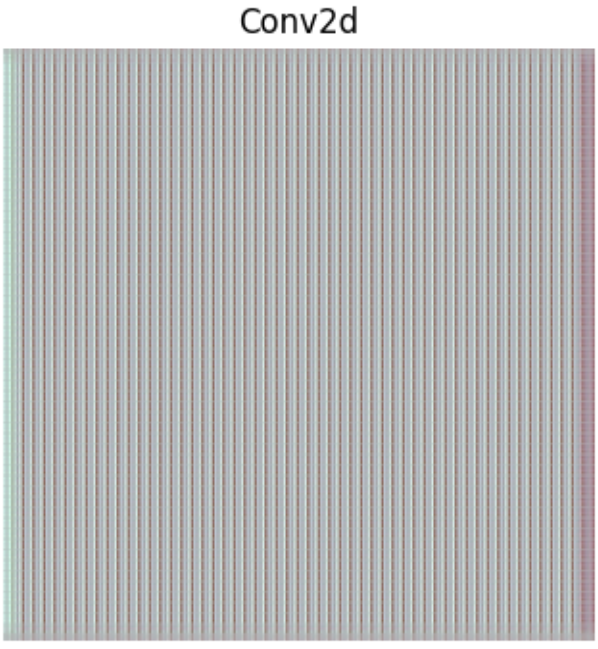
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (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)
  )
)
```

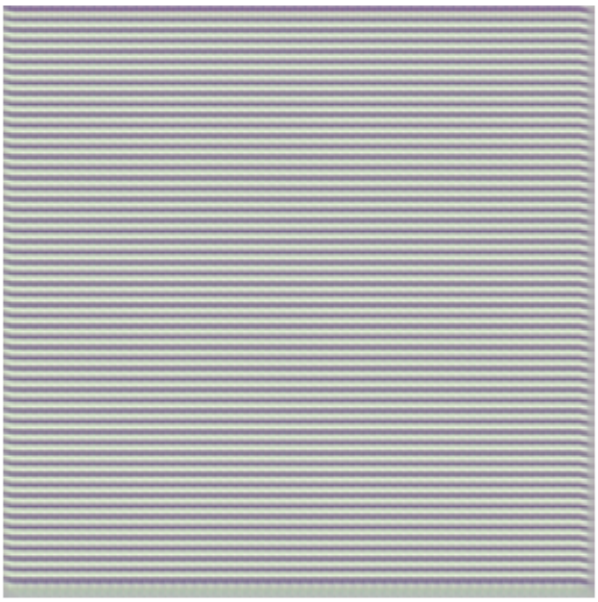
[ ]:
```
#load GradientAscent on GPU
g_ascent = GradientAscent(alexnet.features)
g_ascent.use_gpu = True
```

[ ]:
```
layer_idx = 0

for filters in [5,10,15,20]:
    layer = alexnet.features[layer_idx]
    GradientAscent(alexnet).visualize(layer, filters)
```

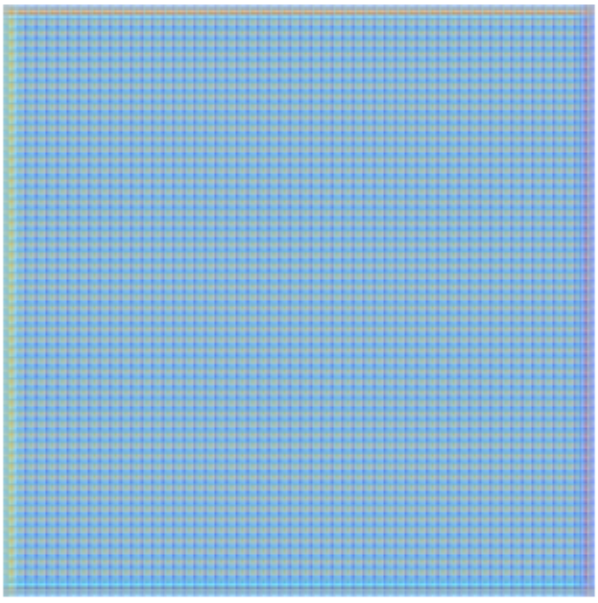
```
/Applications/ANACONDA/anaconda3/envs/py37/lib/python3.7/site-
packages/torch/nn/modules/module.py:795: UserWarning: Using a non-full backward
hook when the forward contains multiple autograd Nodes is deprecated and will be
removed in future versions. This hook will be missing some grad_input. Please
use register_full_backward_hook to get the documented behavior.
  warnings.warn("Using a non-full backward hook when the forward contains
multiple autograd Nodes "
```

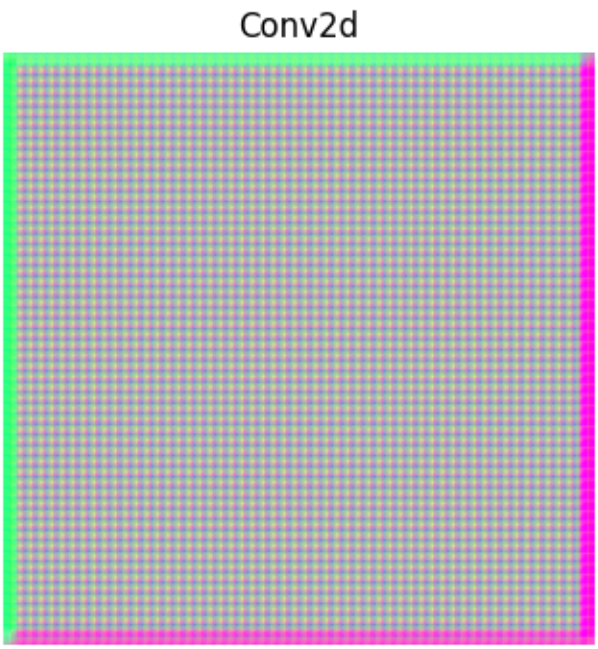


Conv2d



Conv2d

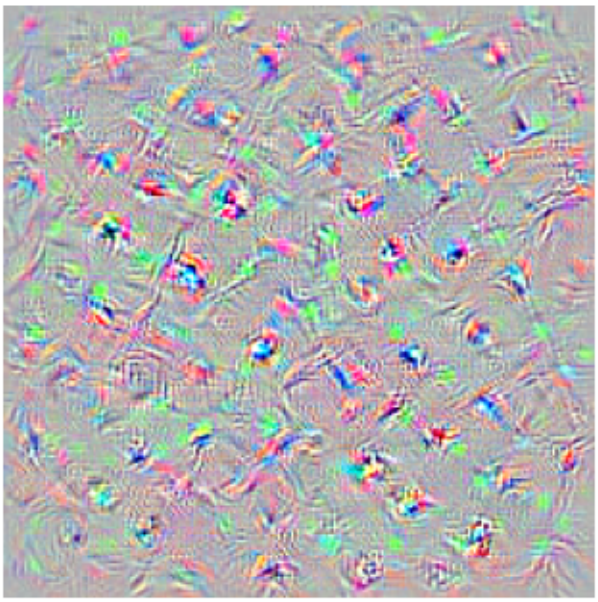




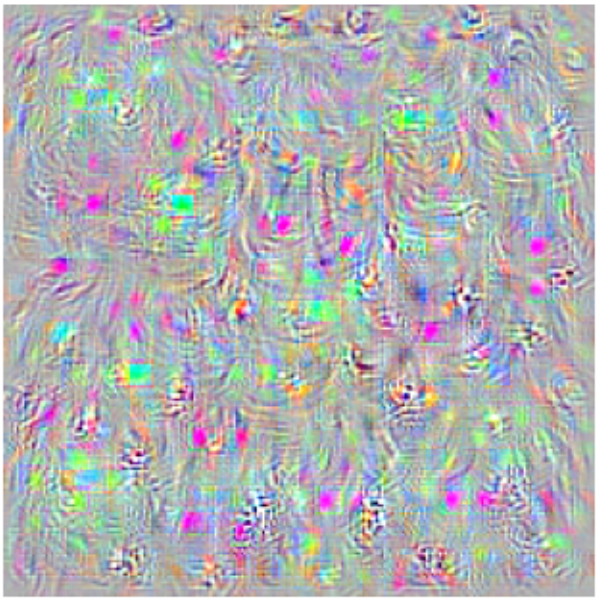
2b: AlexNet layer 10 visualization

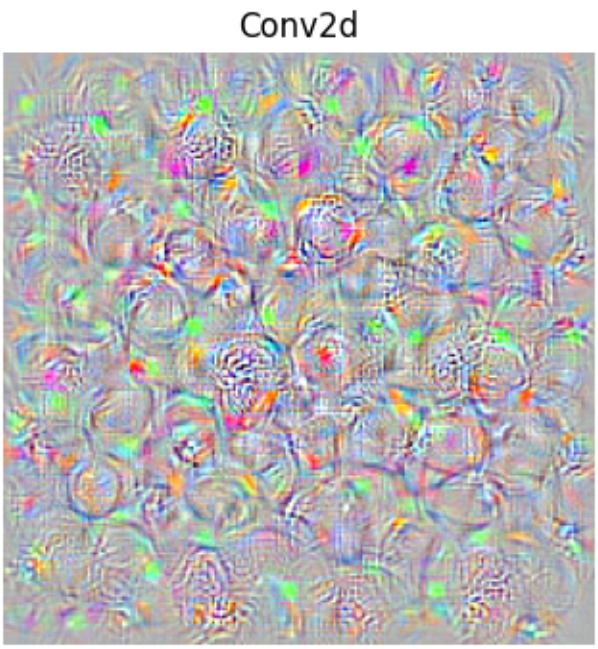
[ ]:
```
layer_idx = 10

for filters in [5,10,15,20]:
    layer = alexnet.features[layer_idx]
    GradientAscent(alexnet).visualize(layer, filters)
```

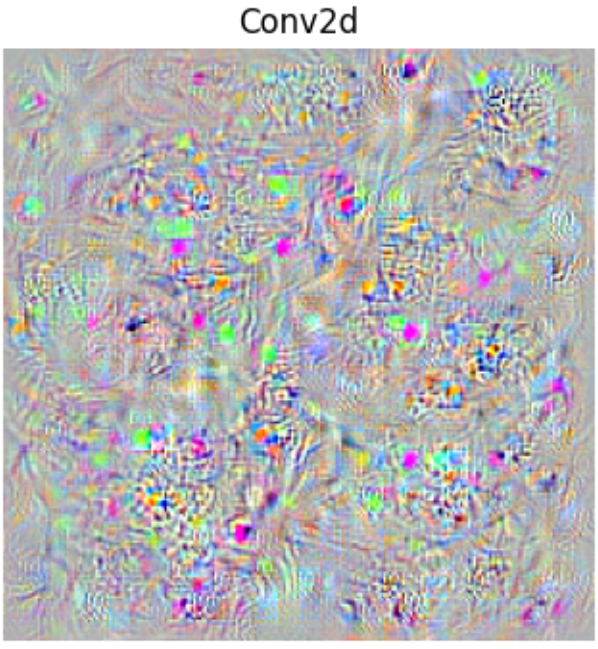
Conv2d



Conv2d





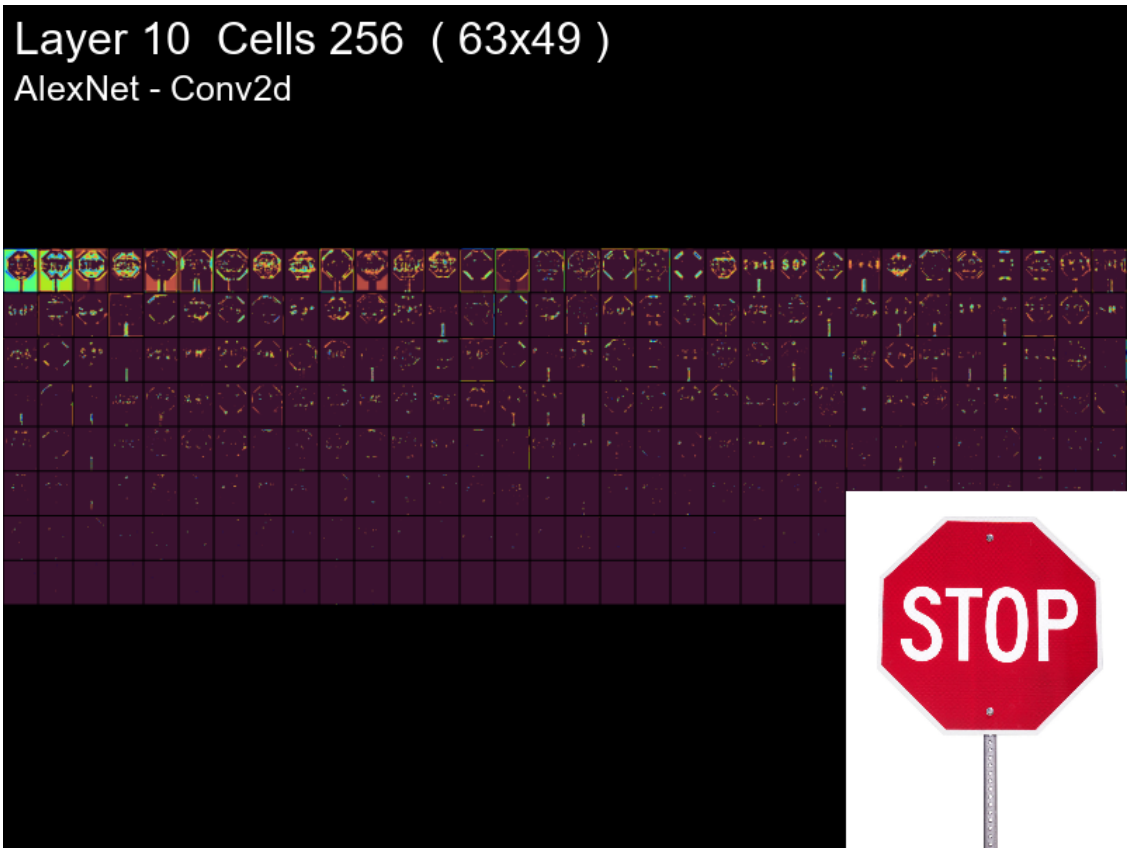


2c: AlexNet saliency map with the stop sign image

[ ]:
```
from MapExtrackt import FeatureExtractor
```

```
layer_idx = 10

fe = FeatureExtractor(alexnet)

fe.set_image(stop_sign_path) # stop_sign_path is the path to the stop sign image

fe.display_from_map(layer_no=layer_idx)
```

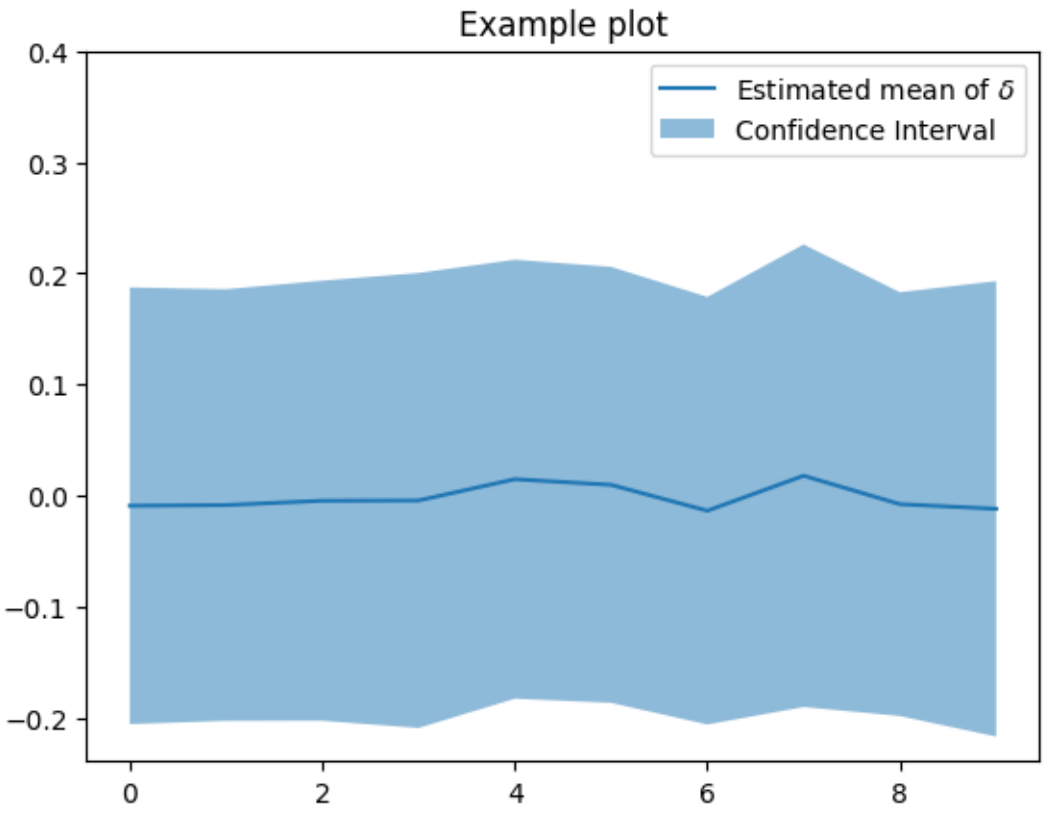
[ ]:



3: Example of plot with a confidence interval

[ ]:
```
#THIS IS JUST AN EXAMPLE TO PLOT CONFIDENCE INTERVAL AS SHADED AREA

n = 500 # number of samples
k = 10  # number of replications
sigma = 0.2

torch.manual_seed(0) # set the random seed
deltas = torch.FloatTensor(sigma*torch.randn(n, k)) # gaussian samples ~ N(0,␣
 ↪sigma*I)

# compute mean and standard deviation
```

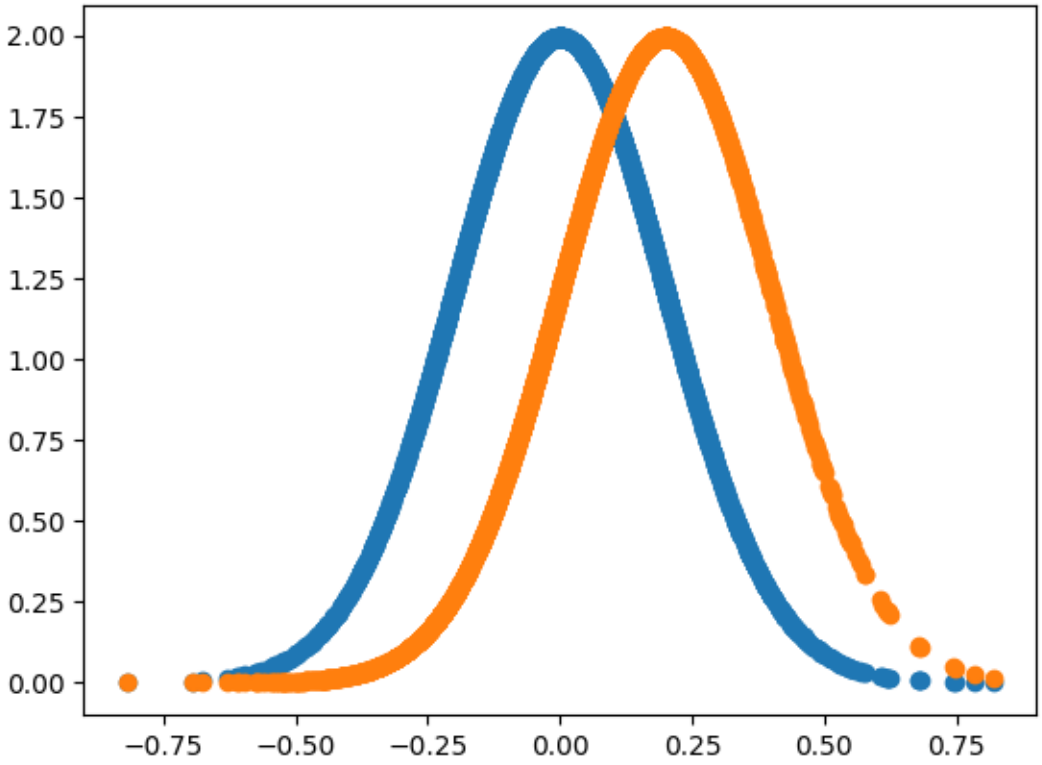
```
mean_ = deltas.mean(dim=0)
std_ = deltas.std(dim=0)

# generate the plot
x = np.arange(k) # populate x axis
plt.plot(x, mean_, label="Estimated mean of $\delta$")
plt.fill_between(x, mean_ - std_, mean_ + std_, alpha=0.5, label="Confidence␣
 ↪Interval") # 1-sigma confidence interval
plt.legend()
plt.ylim([None, 0.4])
plt.title('Example plot')
plt.show()
```



Density computation example with log_prob

[ ]:
```
# THIS IS AN EXAMPLE TO USE log_prob METHOD FOR EASIER DENSITY COMPUTATION

# Suppose you want to compute the density of Normal distribution

# create Normal distribution object
p = Normal(torch.tensor([0.0]), torch.tensor([sigma]))       # N(0, sigma**2)
```

```
p_tilde = Normal(torch.tensor([0.2]), torch.tensor([sigma])) # N(1, sigma**2)

# use log_prob method
log_density_orig = p.log_prob(deltas) # log_prob method gives you log densities
log_density_tilde = p_tilde.log_prob(deltas)

# verify this by plotting the density, i.e. the exp of the log_density
plt.scatter(deltas, torch.exp(log_density_orig), label="p")
plt.scatter(deltas, torch.exp(log_density_tilde), label="p_tilde")
plt.show()
```



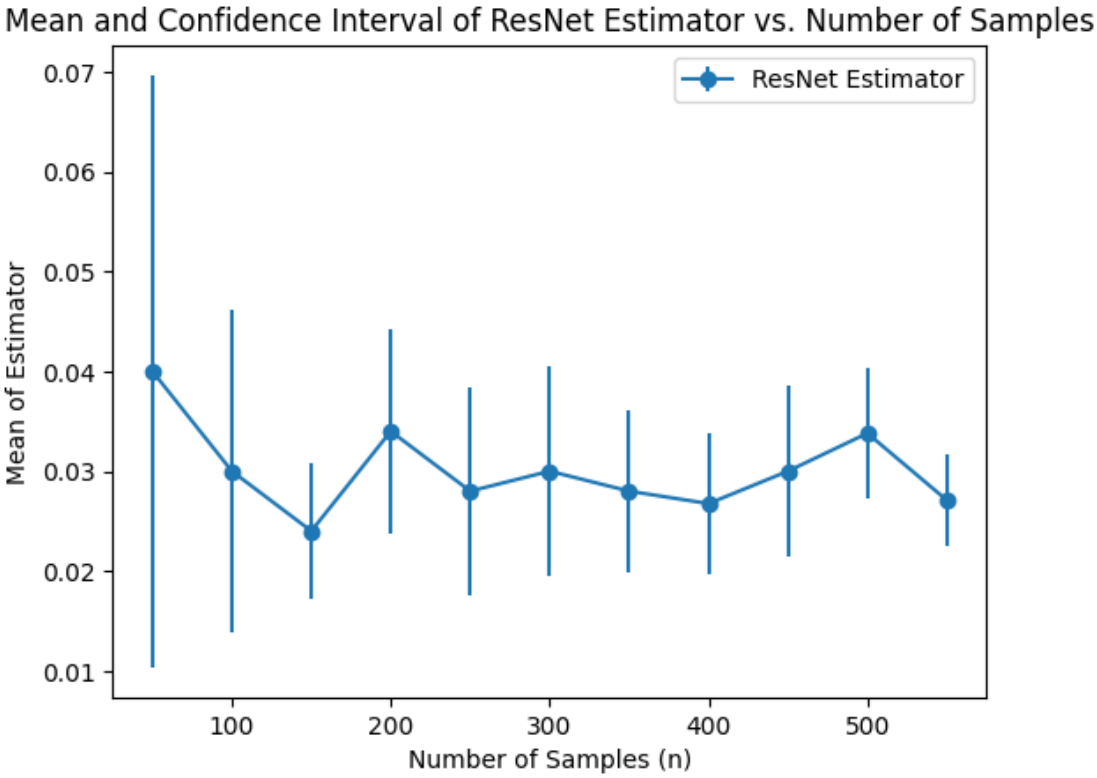
3a: MC estimator for prob. robustness of ResNet-18

[ ]:
```
# REPLACE THE THREE DOTS WITH YOUR OWN CODE

resnet = models.resnet18(pretrained=True).eval()

sigma_squared = 0.2 # parameter sigma

# evaluate the model k times, each time use n samples
k = 10 # number of replications
n_list = list(range(50, 551, 50))  # number of samples in each replication
```

```
# collect the samples
torch.manual_seed(0) # set the random seed

for n in n_list:
    deltas = torch.FloatTensor(sigma_squared*torch.randn(*img.shape, n, k)) #␣
 ↪gaussian samples ~ N(0, sigma**2*I)
    resnet_test=np.zeros([n, k])
    for i in range(k):
        for j in range(n):
            noisy_img = img + deltas[:, :, :, j, i]
            _, _, y_i = get_prediction(noisy_img, resnet)
            resnet_test[j, i] = y_i != true_class
            mu_hat_n_samples = resnet_test.mean(axis=0)
            mean_ = mu_hat_n_samples.mean()
            std_ = mu_hat_n_samples.std()
            mean_, std_
```

[ ]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt

resnet = models.resnet18(pretrained=True).eval()

net = resnet

sigma_squared = 0.2 # parameter sigma

# Number of replications and samples
k = 10 # number of replications
n_values = list(range(50, 551, 50))  # number of samples in each replication
resnet_results = []

for n in n_values:
    torch.manual_seed(0)
    deltas = torch.FloatTensor(sigma_squared * torch.randn(*img.shape, n, k))

    resnet_test = np.zeros([n, k])

    for i in range(k):
        for j in range(n):
            noisy_img = img + deltas[:, :, :, j, i]
            _, _, y_i = get_prediction(noisy_img, net)
            resnet_test[j, i] = y_i != true_class
    misclassification_rate = (resnet_test.sum(axis=0) / n).tolist()
    mu_hat_n_samples = resnet_test.mean(axis=0)
    resnet_results.append(mu_hat_n_samples)
```

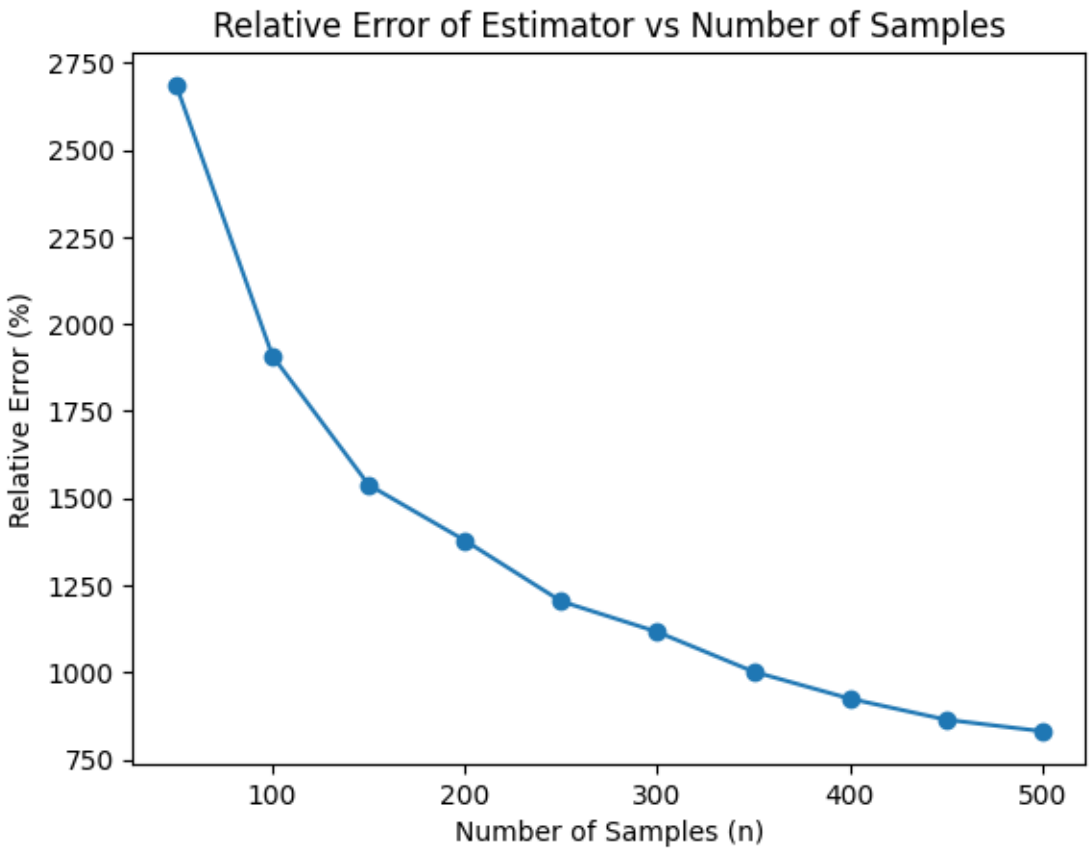
```
# Compute mean and standard deviation for each n
means = [result.mean() for result in resnet_results]
stds = [result.std() for result in resnet_results]

# Plot mean and confidence interval
plt.errorbar(n_values, means, label='ResNet Estimator')
plt.fill_between(n_values, means - stds, means + stds, alpha=0.5,␣
 ↪label="Confidence Interval") # 1-sigma confidence interval
plt.xlabel('Number of Samples (n)')
plt.ylabel('Mean of Estimator')
plt.title('Mean and Confidence Interval of ResNet Estimator vs. Number of␣
 ↪Samples')
plt.legend()
plt.show()
```



3b: MC relative error

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# True value
```

```
mu = 0.03

# Function to compute the relative error
def compute_relative_error(estimator_values, true_value):
    mean_estimator = np.mean(estimator_values)
    std_estimator = np.std(estimator_values, ddof=1)  # ddof=1 for unbiased␣
 ↪estimator of the variance
    relative_error = std_estimator / mean_estimator
    relative_error_percentage = (relative_error / true_value) * 100
    return relative_error_percentage

# Number of replications
k = 1000  # you can adjust this value based on your needs

# List to store relative errors for different n values
relative_errors = []

# Iterate over different sample sizes n
for n in range(50, 501, 50):
    # Generate n i.i.d. samples and compute the estimator
    estimator_values = np.zeros(k)
    for i in range(k):
        deltas = sigma_squared * np.random.randn(n)
        noisy_predictions = np.random.choice([True, False], size=n, p=[mu, 1 -␣
 ↪mu])
        estimator_values[i] = np.mean(noisy_predictions)

    # Compute relative error and store it
    relative_error = compute_relative_error(estimator_values, mu)
    relative_errors.append(relative_error)

# Plot the relative error vs n
plt.plot(range(50, 501, 50), relative_errors, marker='o')
plt.xlabel('Number of Samples (n)')
plt.ylabel('Relative Error (%)')
plt.title('Relative Error of Estimator vs Number of Samples')
plt.show()
```



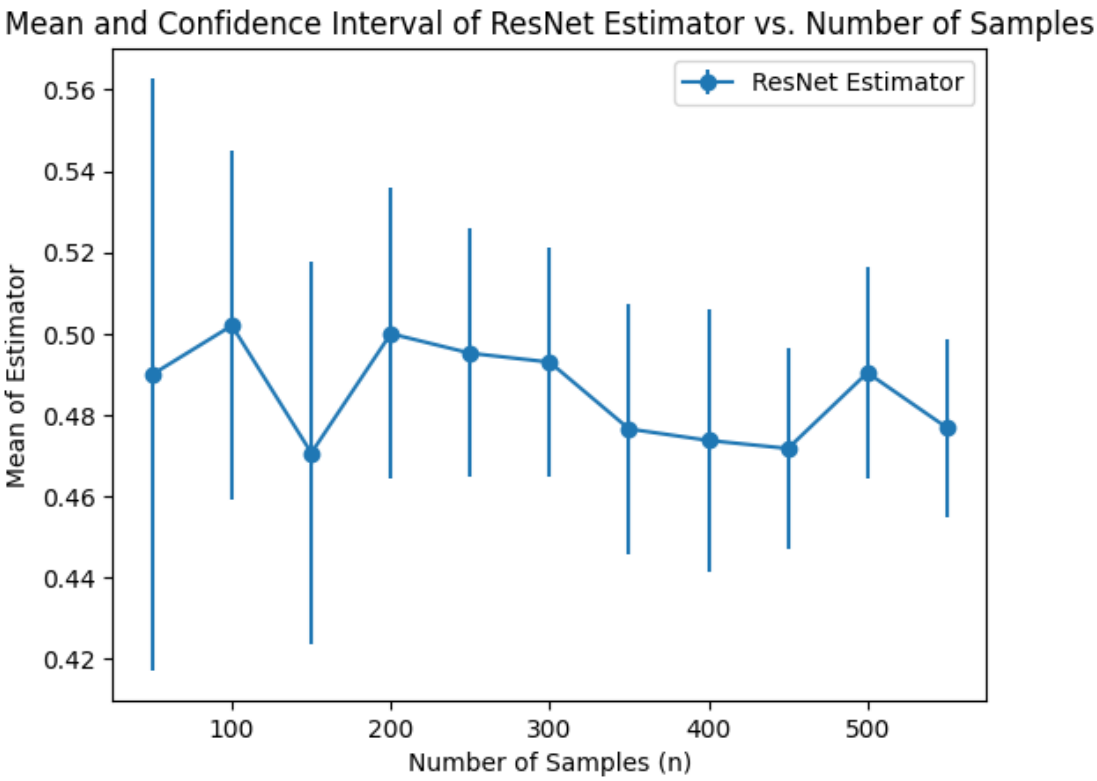
[ ]:

[ ]:

3c: Misclassification rate w.r.t. samples close to an adversarial example

[ ]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt

resnet = models.resnet18(pretrained=True).eval()

net = resnet

sigma_squared = 0.2 # parameter sigma
xtilde= one_call.attack(image_path, atk=attacks.FGM, nstr='resnet18',␣
 ↪ret=True)[0][2]
scale = 1/3
mean_shift = scale * xtilde
```

```
# Number of replications and samples
k = 10 # number of replications
n_values = list(range(50, 551, 50))  # number of samples in each replication
resnet_results = []

for n in n_values:
    torch.manual_seed(0)
    deltas = torch.FloatTensor(mean_shift.unsqueeze(-1).unsqueeze(-1) +␣
 ↪sigma_squared * torch.randn(*img.shape, n, k))

    resnet_test = np.zeros([n, k])

    for i in range(k):
        for j in range(n):
            noisy_img = img + deltas[:, :, :, j, i]
            _, _, y_i = get_prediction(noisy_img, net)
            resnet_test[j, i] = y_i != true_class

    mu_hat_n_samples = resnet_test.mean(axis=0)
    resnet_results.append(mu_hat_n_samples)

# Compute mean and standard deviation for each n
means = [result.mean() for result in resnet_results]
stds = [result.std() for result in resnet_results]

# Plot mean and confidence interval
plt.errorbar(n_values, means, yerr=stds, fmt='-o', label='ResNet Estimator')
plt.xlabel('Number of Samples (n)')
plt.ylabel('Mean of Estimator')
plt.title('Mean and Confidence Interval of ResNet Estimator vs. Number of␣
 ↪Samples')
plt.legend()
plt.show()
```


Mean and Confidence Interval of ResNet Estimator vs. Number of Samples
ResNet Estimator
Mean of Estimator
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.42
100
200
300
400
500
Number of Samples (n)