Prbolem Statement: Take a small image dataset (400-500 images with at least 224X224X3 pixels or more). The dataset must have 4 to 5 classes of images. Find the final features of those images using pre-trained ResNet-101. Take one image from each class, then find the ten nearest neighbors of that image (using the final image embedding) from all the images. Do the same for AlexNet and compare the results.

Group members list

- **Member 1:** M. Zanibul Haque Shanto (1921089042)
- **Member 2:** Md Shawmoon Azad (1912374042)
- **Member 3:** Md. Sajjad Hossain (1922000042)
- **Member 4:** Mohammed Rakibul Hasan (1921798042)
- **Member 5:** Habiba Rashid Lamiya (1922177642)

Summary of the results: As mentioned in the problem statement we have taken a small dog dataset of 455 images which contains 5 classes. Then we used prestarined ResNet101 to extract the final features. Then we have used image embedding to find the 10 nearest images. we have taken help from Deeplake'Hub' for the final image embedding. Using the Hub API the final result we have go is that for 1 chihuahua image we have got 10 similar images of chihuahua as a result.

Installing Libraires

```
[ ] !pip install hub
    !pip install torch
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting hub
  Downloading hub-3.0.1-py3-none-any.whl (1.4 kB)
Collecting deeplake
  Downloading deeplake-3.1.5.tar.gz (404 kB)
     |████████████████████████████████| 404 kB 5.3 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from deeplake->hub) (1.21.6)
Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-packages (from deeplake->hub) (7.1.2)
Collecting boto3
  Downloading boto3-1.26.37-py3-none-any.whl (132 kB)
     |████████████████████████████████| 132 kB 49.0 MB/s
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from deeplake->hub) (7.1.2)
Collecting pathos
  Downloading pathos-0.3.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 7.4 MB/s
Collecting humbug>=0.2.6
  Downloading humbug-0.2.7-py3-none-any.whl (11 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from deeplake->hub) (4.64.1)
Collecting numcodecs
  Downloading numcodecs-0.11.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (6.7 MB)
     |████████████████████████████████| 6.7 MB 39.2 MB/s
Collecting pyjwt
  Downloading PyJWT-2.6.0-py3-none-any.whl (20 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from humbug>=0.2.6->deeplake->hub) (2.23.0)
Collecting s3transfer<0.7.0,>=0.6.0
  Downloading s3transfer-0.6.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 4.2 MB/s
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting botocore<1.30.0,>=1.29.37
  Downloading botocore-1.29.37-py3-none-any.whl (10.3 MB)
     |████████████████████████████████| 10.3 MB 34.3 MB/s
Collecting urllib3<1.27,>=1.25.4
  Downloading urllib3-1.26.13-py2.py3-none-any.whl (140 kB)
     |████████████████████████████████| 140 kB 44.4 MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.8/dist-packages (from botocore<1.30.0,>=1.29.37->boto3->deeplake-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.30.0,>=1.29.37->boto3-
Requirement already satisfied: entrypoints in /usr/local/lib/python3.8/dist-packages (from numcodecs->deeplake->hub) (0.4)
Collecting pox>=0.3.2
  Downloading pox-0.3.2-py3-none-any.whl (29 kB)
Collecting ppft>=1.7.6.6
  Downloading ppft-1.7.6.6-py3-none-any.whl (52 kB)
     |████████████████████████████████| 52 kB 685 kB/s
Collecting multiprocess>=0.70.14
  Downloading multiprocess-0.70.14-py38-none-any.whl (132 kB)
     |████████████████████████████████| 132 kB 44.2 MB/s
Requirement already satisfied: dill>=0.3.6 in /usr/local/lib/python3.8/dist-packages (from pathos->deeplake->hub) (0.3.6)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->humbug>=0.2.6->deeplake->hub) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->humbug>=0.2.6->deeplake->hub) (2022.12.7)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->humbug>=0.2.6->deeplake->hub) (3.0.4)
Collecting urllib3<1.27,>=1.25.4
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
     |████████████████████████████████| 127 kB 44.4 MB/s
Building wheels for collected packages: deeplake
  Building wheel for deeplake (setup.py) ... done
  Created wheel for deeplake: filename=deeplake-3.1.5-py3-none-any.whl size=489424 sha256=8f4a81158cbc9e7c3623585017b93ea3940f295d9248e27d9e9e57d995bd
```

```
  Stored in directory: /root/.cache/pip/wheels/e7/ca/54/d118b254f85536e95e27056f1381b7bd43a2f46b208869cf66
```

Importing Libraries

```
[ ] import numpy as np
    import hub
    from torch.utils.data import DataLoader
    from torchvision import datasets, transforms
    import torch
    from torchsummary import summary
    import torchvision.models as models
    import glob
    from tqdm import tqdm
    from PIL import Image

    import matplotlib.pyplot as plt
    %matplotlib inline
```

Loading dataset

Loading Dataset from the GDrive

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

Exploratory Data Analysis

```
[ ] # Viewing the dataset folder
    !ls /content/drive/MyDrive/data/Dataset_Dogs
```

```
    chihuahua  corgi  dachshund  labrador  pug
```

```
[ ] data_dir = '/content/drive/MyDrive/data/Dataset_Dogs'
```

```
[ ] list_imgs = glob.glob(data_dir + "/**/*.jpg")
    print(f"There are {len(list_imgs)} images in the dataset {data_dir}")
```

```
    There are 455 images in the dataset /content/drive/MyDrive/data/Dataset_Dogs
```

```
[ ] # create dataloader with required transforms
    tc = transforms.Compose([
            transforms.Resize((256, 256)),
            transforms.ToTensor()
        ])

    image_datasets = datasets.ImageFolder(data_dir, transform=tc)
    dloader = torch.utils.data.DataLoader(image_datasets, batch_size=10, shuffle=False)
```

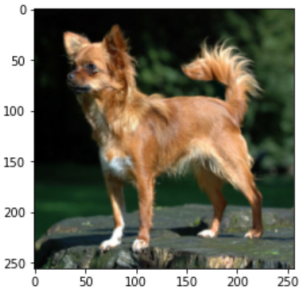
```
[ ] for img, label in dloader:
      print(np.transpose(img[0], (1,2,0)).shape)
      print(img[0])
      plt.imshow((img[0].detach().numpy().transpose(1, 2, 0)*255).astype(np.uint8))
      plt.show()
      break
```

```
    torch.Size([256, 256, 3])
    tensor([[[0.0118, 0.0157, 0.0157,  ..., 0.1882, 0.1922, 0.1922],
             [0.0118, 0.0157, 0.0157,  ..., 0.1922, 0.1961, 0.1961],
             [0.0118, 0.0157, 0.0196,  ..., 0.1961, 0.1961, 0.1961],
             ...,
             [0.0392, 0.0353, 0.0392,  ..., 0.4314, 0.3725, 0.3569],
             [0.0314, 0.0392, 0.0392,  ..., 0.5529, 0.4275, 0.3922],
             [0.0275, 0.0353, 0.0196,  ..., 0.5255, 0.4392, 0.3843]],

            [[0.0118, 0.0157, 0.0157,  ..., 0.2902, 0.2941, 0.2941],
             [0.0118, 0.0157, 0.0157,  ..., 0.2941, 0.2980, 0.2980],
             [0.0118, 0.0157, 0.0196,  ..., 0.2941, 0.2941, 0.2941],
             ...,
             [0.0627, 0.0627, 0.0627,  ..., 0.4510, 0.3765, 0.3451],
             [0.0588, 0.0549, 0.0667,  ..., 0.5412, 0.4118, 0.3843],
             [0.0549, 0.0471, 0.0549,  ..., 0.5098, 0.4196, 0.3882]],

            [[0.0118, 0.0157, 0.0157,  ..., 0.1490, 0.1529, 0.1451],
             [0.0118, 0.0157, 0.0157,  ..., 0.1569, 0.1608, 0.1529],
             [0.0118, 0.0157, 0.0196,  ..., 0.1647, 0.1686, 0.1608],
             ....
```

```
        [0.0510, 0.0549, 0.0510,  ..., 0.3922, 0.3255, 0.2902],
        [0.0471, 0.0549, 0.0549,  ..., 0.5098, 0.3451, 0.3137],
        [0.0549, 0.0627, 0.0392,  ..., 0.4980, 0.3569, 0.3059]]])
```



Pytorch default backend for images are Pillow, and when you use ToTensor()class, PyTorch automatically converts all images into [0,1] so no need to normalize the images here.

```
[ ]  len(image_datasets)
```

```
455
```

Generate embeddings

```
[ ]  def copy_embeddings(m, i, o):
         """Copy embeddings from the penultimate layer.
         """
         o = o[:, :, 0, 0].detach().numpy().tolist()
         outputs.append(o)
```

Fetching Pretrained Resnet101 model

```
[ ]
     model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet101', pretrained=True)
```

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.cache/torch/hub/v0.10.0.zip
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be rem
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are dep
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet101-63fe2227.pth" to /root/.cache/torch/hub/checkpoints/resnet101-63fe2227.pth
100% 171M/171M [00:01<00:00, 174MB/s]
```

```
[ ]
     layer = model._modules.get('avgpool')

     _ = layer.register_forward_hook(copy_embeddings)
```

```
[ ]  outputs = []

     model.eval()
```

```
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (10): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (11): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (12): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (13): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (14): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (15): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (16): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

Generating Image Embaddings

```
[ ] for X, y in dloader:
        _ = model(X)
```

```
/usr/local/lib/python3.8/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))
```

```
[ ] len(outputs)
```

```
46
```

```
[ ] # flatten list of embeddings to remove batches
    list_embeddings = [item for sublist in outputs for item in sublist]
    print(len(list_embeddings))
```

```
455
```

```
[ ] assert len(list_embeddings) == len(image_datasets)
```

```
[ ] np.array(list_embeddings[0]).shape
```

```
(2048,)
```

```
[ ] #Sending to Hub using API
    !activeloop login -u shawmoon -p activeloop786
```

```
Successfully logged in to Activeloop.
```

```
[ ] #Creating a New Hub folder
    with hub.empty('./dog_deeplake') as ds:
        # Create the tensors
        ds.create_tensor('images', htype = 'image', sample_compression = 'jpeg')
        ds.create_tensor('embeddings')


        ds.info.update(description = 'Dog breeds embeddings dataset')
        ds.images.info.update(camera_type = 'SLR')

        # Iterate through the images and their corresponding embeddings, and append them to hub dataset
        for i in tqdm(range(len(image_datasets))):
          img = image_datasets[i][0].detach().numpy().transpose(1, 2, 0)
          img = img * 255 # images are normalized
          img = img.astype(np.uint8)
          ds.images.append(img)  # Append to Hub Dataset
          ds.embeddings.append(list_embeddings[i]) # Append to Hub Dataset
```

```
100%|██████████| 455/455 [00:15<00:00, 29.17it/s]
```

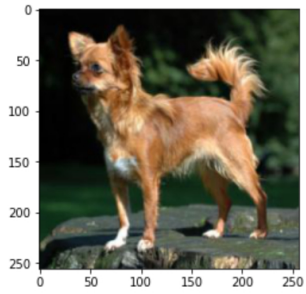
```
[ ]
```

Output of 10 nearest images of the class

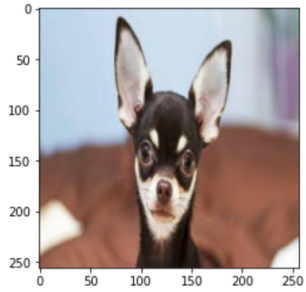
```
[ ]  def show_image_in_ds(ds, idx=1):
         image = ds.images[idx].numpy()
         embedding = ds.embeddings[idx].numpy()
         print("Image:")
         print(image.shape)
         plt.imshow(image)
         plt.show()
         print(embedding[0:10]) # show only 10 first values of the image embedding

     for i in range(10):
         show_image_in_ds(ds, i)
```

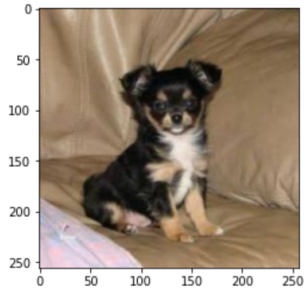
```
Image:
(256, 256, 3)
```




```
[0.10145672 0.70408124 0.75996172 0.07339834 0.15769057 0.05572629
 0.42745185 0.19763529 0.30676952 0.10714862]
Image:
(256, 256, 3)
```




```
[0.26521778 1.53198338 0.84992194 0.23684855 0.14586289 0.11369801
 0.45144984 0.09844674 0.76923048 0.18753363]
Image:
(256, 256, 3)
```




```
[0.25586802 1.03631294 0.93826485 0.21968383 0.10480054 0.1986323
 0.6581403  0.04625438 0.02174159 0.14413106]
Image:
(256, 256, 3)
```

Doing Simillar For AlexNet as well

```
[ ]  # create dataloader with required transforms
     tc = transforms.Compose([
             transforms.Resize((227, 227)),
             transforms.ToTensor()
         ])

     image_datasets = datasets.ImageFolder(data_dir, transform=tc)
     dloader = torch.utils.data.DataLoader(image_datasets, batch_size=10, shuffle=False)
```

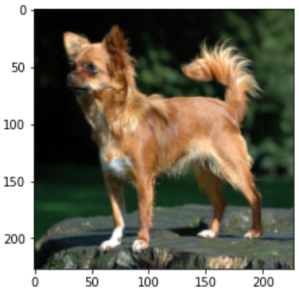
```
[ ]  for img, label in dloader:
       print(np.transpose(img[0], (1,2,0)).shape)
       print(img[0])
       plt.imshow((img[0].detach().numpy().transpose(1, 2, 0)*255).astype(np.uint8))
```

```
    plt.show()
    break
```

```
torch.Size([227, 227, 3])
tensor([[[0.0118, 0.0157, 0.0157,  ..., 0.1922, 0.1922, 0.1922],
         [0.0118, 0.0157, 0.0196,  ..., 0.1882, 0.1961, 0.1922],
         [0.0118, 0.0157, 0.0196,  ..., 0.1922, 0.1961, 0.1961],
         ...,
         [0.0392, 0.0353, 0.0392,  ..., 0.3961, 0.3765, 0.3569],
         [0.0314, 0.0392, 0.0353,  ..., 0.4824, 0.4510, 0.3882],
         [0.0314, 0.0314, 0.0157,  ..., 0.4745, 0.4627, 0.3882]],

        [[0.0118, 0.0157, 0.0157,  ..., 0.2941, 0.2941, 0.2941],
         [0.0118, 0.0157, 0.0196,  ..., 0.2902, 0.2980, 0.2941],
         [0.0118, 0.0157, 0.0196,  ..., 0.2902, 0.2941, 0.2941],
         ...,
         [0.0667, 0.0627, 0.0588,  ..., 0.4235, 0.3804, 0.3451],
         [0.0588, 0.0588, 0.0588,  ..., 0.4902, 0.4431, 0.3804],
         [0.0549, 0.0471, 0.0549,  ..., 0.4627, 0.4471, 0.3922]],

        [[0.0118, 0.0157, 0.0157,  ..., 0.1529, 0.1529, 0.1490],
         [0.0118, 0.0157, 0.0196,  ..., 0.1569, 0.1608, 0.1569],
         [0.0118, 0.0157, 0.0196,  ..., 0.1725, 0.1725, 0.1647],
         ...,
         [0.0588, 0.0588, 0.0549,  ..., 0.3529, 0.3255, 0.2941],
         [0.0471, 0.0510, 0.0431,  ..., 0.4588, 0.3804, 0.3098],
         [0.0549, 0.0588, 0.0314,  ..., 0.4588, 0.3922, 0.3098]]])
```




```
[ ] def copy_embeddings(m, i, o):
        """Copy embeddings from the penultimate layer.
        """
        o = o[:, :, 0, 0].detach().numpy().tolist()
        outputs.append(o)
```

```
[ ] model = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet', pretrained=True)
```

```
Using cache found in /root/.cache/torch/hub/pytorch_vision_v0.10.0
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are dep
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100% 233M/233M [00:01<00:00, 141MB/s]
```

```
[ ]
    layer = model._modules.get('avgpool')
    _ = layer.register_forward_hook(copy_embeddings)

    outputs = []

    model.eval()
```

```
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)
  )
)
```

```
,
```

```
[ ] # Generate image's embeddings for all images in dloader and saves
    for X, y in dloader:
        _ = model(X)
```

```
[ ] # flatten list of embeddings to remove batches
    list_embeddings = [item for sublist in outputs for item in sublist]
    print(len(list_embeddings))
```

```
455
```

```
[ ] assert len(list_embeddings) == len(image_datasets)
```

```
[ ] np.array(list_embeddings[0]).shape
```

```
(256,)
```

```
[ ] !activeloop login -u shawmoon -p activeloop786
```

```
Successfully logged in to Activeloop.
```

```
[ ] #Creating a New Hub folder
    with hub.empty('./dog2_deeplake') as ds:

        ds.create_tensor('images', htype = 'image', sample_compression = 'jpeg')
        ds.create_tensor('embeddings')

        ds.info.update(description = 'Dog breeds embeddings dataset')
        ds.images.info.update(camera_type = 'SLR')


        for i in tqdm(range(len(image_datasets))):
          img = image_datasets[i][0].detach().numpy().transpose(1, 2, 0)
          img = img * 255
          img = img.astype(np.uint8)
          ds.images.append(img)
          ds.embeddings.append(list_embeddings[i])
```

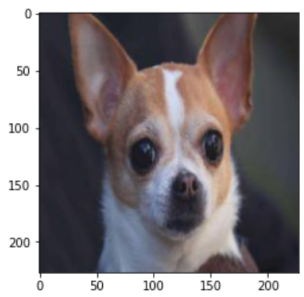
```
100%|██████████| 455/455 [00:13<00:00, 34.14it/s]
```

Final Output of the 10 nearest neighbor images of the given Image using Image Embedding

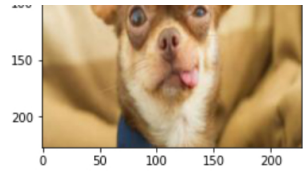
```
[ ] def show_image_in_ds(ds, idx=1):
        image = ds.images[idx].numpy()
        embedding = ds.embeddings[idx].numpy()
        print("Image:")
        print(image.shape)
        plt.imshow(image)
        plt.show()
        print(embedding[0:10]) # show only 10 first values of the image embedding

    for i in range(10):
        show_image_in_ds(ds, i)
```

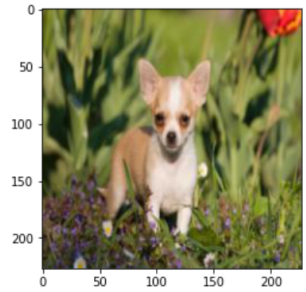
```
 0.         0.         0.         0.         ]
Image:
(227, 227, 3)
```




```
[0.         0.         0.         0.         0.
 0.41105703 0.34872171 0.         0.         ]
Image:
(227, 227, 3)
```




```
[0.         0.         0.         0.         0.         1.30156076
 3.25952077 0.         0.         0.        ]
Image:
(227, 227, 3)
```




```
[0.         0.13351397 0.         0.02277862 0.         0.
 0.         0.         1.21417987 0.        ]
```