Install easy-vqa library

```
from google.colab import drive

drive.mount("/content/drive", force_remount=True)
```

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

```
import os

os.chdir(
    "/content/drive/MyDrive/VQA"
)  # fill in with the path to the google drive folder where your mp is.
```

```
# this package has all the data needed to train our model
!pip install easy-vqa
```

```
Requirement already satisfied: easy-vqa in /usr/local/lib/python3.10/dist-packages (1.0)
```

Inputs to model

We would need the following inputs to our model in the form of

- `[Image, Question, Answer]`

For train and test datasets these will be the variable inputs

- `[train_X_ims, train_X_seqs, train_Y]` - train image-question-answer input
- `[test_X_ims, test_X_seqs, test_Y]` - test image-question-answer input

We would also need a

- embedding_size - size of embedding for input node count
- im_shape - shape of image for input node count
- num_answers - number of answers for output layer node count

Loading and preprocessing Images

```
import numpy as np
import torch
import torchvision
import torchvision.transforms as T
from torchvision.transforms import functional as F
from PIL import Image
from easy_vqa import get_train_questions, get_test_questions, get_train_image_paths, get_test_image_paths, get_answers

def load_and_process_image(image_path):
    # Loads image from path and converts to Tensor, you can also reshape the im
    im = Image.open(image_path)
    im = F.to_tensor(im)
    return im

def read_images(paths):
    # paths is a dict mapping image ID to image path
    # Returns a dict mapping image ID to the processed image
    ims = {}
    for image_id, image_path in paths.items():
        ims[image_id] = load_and_process_image(image_path)
    return ims

print('--- Reading/processing images from image paths of the vqa library ---\n')
train_ims = read_images(get_train_image_paths())
test_ims = read_images(get_test_image_paths())
im_shape = train_ims[0].shape
print(f'Read {len(train_ims)} training images and {len(test_ims)} testing images.')
print(f'Each image has shape {im_shape}.')

print('\n--- Creating model input images...')
train_qs, train_answers, train_image_ids = get_train_questions()
test_qs, test_answers, test_image_ids = get_test_questions()
train_X_ims = np.array([train_ims[id] for id in train_image_ids])
test_X_ims = np.array([test_ims[id] for id in test_image_ids])
```

```
--- Reading/processing images from image paths of the vqa library ---

Read 4000 training images and 1000 testing images.
Each image has shape torch.Size([3, 64, 64]).

--- Creating model input images...
<ipython-input-7-c6865f93d0ef>:33: FutureWarning: The input object of type 'Tensor' is an array-like implementing one of
  train_X_ims = np.array([train_ims[id] for id in train_image_ids])
<ipython-input-7-c6865f93d0ef>:33: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is
  train_X_ims = np.array([train_ims[id] for id in train_image_ids])
<ipython-input-7-c6865f93d0ef>:34: FutureWarning: The input object of type 'Tensor' is an array-like implementing one of
  test_X_ims = np.array([test_ims[id] for id in test_image_ids])
<ipython-input-7-c6865f93d0ef>:34: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is
  test_X_ims = np.array([test_ims[id] for id in test_image_ids])
```

Loading Questions and Answers

```
print('\n--- Reading questions...')
train_qs, train_answers, train_image_ids = get_train_questions()
test_qs, test_answers, test_image_ids = get_test_questions()
print(f'Read {len(train_qs)} training questions and {len(test_qs)} testing questions.')

print('\n--- Reading answers...')
all_answers = get_answers()
num_answers = len(all_answers)
print(f'Found {num_answers} total answers:')
print(all_answers)
```

```
--- Reading questions...
Read 38575 training questions and 9673 testing questions.

--- Reading answers...
Found 13 total answers:
['circle', 'green', 'red', 'gray', 'yes', 'teal', 'black', 'rectangle', 'yellow', 'triangle', 'brown', 'blue', 'no']
```

Quick look at the dataset

```
import pandas as pd
df = pd.DataFrame(list(zip(train_qs, train_answers, train_image_ids)), columns =['Question', 'Answer', 'Image ID'])
df.head(10)
```

	Question	Answer	Image ID
0	what is the blue shape?	rectangle	0
1	what color is the shape?	blue	0
2	does the image contain a rectangle?	yes	0
3	is there a triangle in the image?	no	0
4	is there a black shape?	no	0
5	does the image not contain a gray shape?	yes	0
6	is there a red shape in the image?	no	0
7	does the image not contain a red shape?	yes	0
8	is there not a blue shape?	no	0
9	is there not a blue shape in the image?	no	0

Quick look at images

```
import torchvision.utils as utils
from torchvision import transforms

# print multiple images
# images = 1
# batch = torch.empty((images, 3, 64, 64))
# for i in range(images):
#     batch[i] = train_ims[i]

# Create a grid of images
id = 0
grid = utils.make_grid(train_ims[id], nrow=2)
# Convert the grid to a PIL image
```

```
image = transforms.ToPILImage()(grid)
# Show the image
image.show()
```

Preprocessing Questions

```
! pip install -U sentence-transformers
from sentence_transformers import SentenceTransformer, util
st_model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1')

#Questions are encoded by calling model.encode()
train_X_seqs = st_model.encode(train_qs)
test_X_seqs = st_model.encode(test_qs)

# convert ndarray to tensor
train_X_seqs = torch.tensor(train_X_seqs, dtype=torch.float)
test_X_seqs = torch.tensor(test_X_seqs, dtype=torch.float)

print(f'\nThe shape of the binary vectors is : {train_X_seqs.shape}')
```

Preprocessing Answers

```
print('\n--- Creating model outputs...')

train_answer_indices = np.array([all_answers.index(a) for a in train_answers])
test_answer_indices = np.array([all_answers.index(a) for a in test_answers])

#creating a 2D array filled with 0's
train_Y = np.zeros((train_answer_indices.size, train_answer_indices.max()+1), dtype=int)
test_Y = np.zeros((test_answer_indices.size, test_answer_indices.max()+1), dtype=int)

#replacing 0 with a 1 at the index of the original array
train_Y[np.arange(train_answer_indices.size),train_answer_indices] = 1
test_Y[np.arange(test_answer_indices.size),test_answer_indices] = 1

# finally convert the label vectors to tensor and fix the data type so it wouldnt error in the fully connected layer
train_Y = torch.tensor(train_Y, dtype=torch.float)
test_Y = torch.tensor(test_Y, dtype=torch.float)

print(f'Example model output: {train_Y[0]}')
print(f'data type {type(train_Y)}')
```

```
--- Creating model outputs...
Example model output: tensor([0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.])
data type <class 'torch.Tensor'>
```

```
  Building wheel for sentence-transformers (setup.py) ... done
```

The Model

```
from torchvision.models.vgg import vgg19
import torch
import torchvision
from torch import mul, cat, tanh, relu

class VQA_v2(torch.nn.Module):
  def __init__(self, embedding_size, num_answers):
    super(VQA_v2, self).__init__()

    # The Image network which processes image and outputs a vector of shape (batch_size x 32)
    vgg = vgg19(pretrained=True)
    self.features = vgg.features


    self.avgpool = vgg.avgpool
    self.classifier = torch.nn.Sequential(
      torch.nn.Linear(25088, 512),
      torch.nn.Tanh(),
      torch.nn.Linear(512, 128),
      torch.nn.Tanh(),
      torch.nn.Linear(128, 32)
        )


    # The question network processes the question and outputs a vector of shape (batch_size x 32)
    self.fc2 = torch.nn.Linear(embedding_size, 64)      # (384, 64)
    self.fc3 = torch.nn.Linear(64, 32)                  # (64, 32)

    # Layers for Merging operation
    self.fc4 = torch.nn.Linear(64, 32)
```

```
    self.fc5 = torch.nn.Linear(32, num_answers)

  def forward(self, x, q):
    # The Image network
    x = self.features(x)                              # (batch_size, 32)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.classifier(x)

    # The question network
    act = torch.nn.Tanh()
    q = act(self.fc2(q))                        # (32, 32)
    q = act(self.fc3(q))                        # (32, 32)

    # Merge -> output
    out = cat((x, q), 1)                        # concat function
    out = act(self.fc4(out))                    # activation
    out = self.fc5(out)                         # output probability
    return out
```

Custom Dataset

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self, img, txt, ans):
      self.img = img
      self.txt = txt
      self.ans = ans

    def __len__(self):
        return len(self.ans)

    def __getitem__(self, idx):
      ans = self.ans[idx]
      img = self.img[idx]
      txt = self.txt[idx]
      return img, txt, ans
```

Train and evaluate loops

```
def train_loop(model, optimizer, criterion, train_loader):
    model.train()
    model.to(device)
    total_loss, total = 0, 0

    for image, text, label in trainloader:
        # get the inputs; data is a list of [inputs, labels]
        image, text, label =  image.to(device), text.to(device), label.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        output = model.forward(image, text)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()

        # Record metrics
        total_loss += loss.item()
        total += len(label)

    return total_loss / total


def validate_loop(model, criterion, valid_loader):
    model.eval()
    model.to(device)
    total_loss, total = 0, 0

    with torch.no_grad():
      for image, text, label in testloader:
          # get the inputs; data is a list of [inputs, labels]
          image, text, label =  image.to(device), text.to(device), label.to(device)

          # Forward pass
          output = model.forward(image, text)
```

```
            output = model.forward(image, text)

            # Calculate how wrong the model is
            loss = criterion(output, label)

            # Record metrics
            total_loss += loss.item()
            total += len(label)

    return total_loss / total
```

WandB for Logging

```
!pip install WandB
import wandb
# login with your API key from wandb account
!wandb login --relogin
```

```
Collecting WandB
  Downloading wandb-0.15.12-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 8.5 MB/s eta 0:00:00
Requirement already satisfied: Click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-packages (from WandB) (8.1.7)
Collecting GitPython!=3.1.29,>=1.0.0 (from WandB)
  Downloading GitPython-3.1.40-py3-none-any.whl (190 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.6/190.6 kB 11.2 MB/s eta 0:00:00
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (2.31.0)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (5.9.5)
Collecting sentry-sdk>=1.0.0 (from WandB)
  Downloading sentry_sdk-1.32.0-py2.py3-none-any.whl (240 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 241.0/241.0 kB 12.1 MB/s eta 0:00:00
Collecting docker-pycreds>=0.4.0 (from WandB)
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from WandB) (6.0.1)
Collecting pathtools (from WandB)
  Downloading pathtools-0.1.2.tar.gz (11 kB)
  Preparing metadata (setup.py) ... done
Collecting setproctitle (from WandB)
  Downloading setproctitle-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from WandB) (67.7.2)
Requirement already satisfied: appdirs>=1.4.3 in /usr/local/lib/python3.10/dist-packages (from WandB) (1.4.4)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.19.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (3.2
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from docker-pycreds>=0.4.0->WandB)
Collecting gitdb<5,>=4.0.1 (from GitPython!=3.1.29,>=1.0.0->WandB)
  Downloading gitdb-4.0.11-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 8.9 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->WandB)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->W
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->W
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->GitPython!=3.1.29,>=1.0.0->WandB)
  Downloading smmap-5.0.1-py3-none-any.whl (24 kB)
Building wheels for collected packages: pathtools
  Building wheel for pathtools (setup.py) ... done
  Created wheel for pathtools: filename=pathtools-0.1.2-py3-none-any.whl size=8791 sha256=42a4e50bc40b5a127e2ff438cd9f4e
  Stored in directory: /root/.cache/pip/wheels/e7/f3/22/152153d6eb222ee7a56ff8617d80ee5207207a8c00a7aab794
Successfully built pathtools
Installing collected packages: pathtools, smmap, setproctitle, sentry-sdk, docker-pycreds, gitdb, GitPython, WandB
Successfully installed GitPython-3.1.40 WandB-0.15.12 docker-pycreds-0.4.0 gitdb-4.0.11 pathtools-0.1.2 sentry-sdk-1.32.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

```
# set path
from pathlib import Path
from google.colab import drive
drive.mount('/content/drive')
project_path = '/content/drive/MyDrive/FSDL/'
project_path = Path(project_path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

Training Run (+ Dataloaders and Hyper parameters)

```
from torch.utils.data import DataLoader

from tqdm.notebook import tqdm
# WandB – Initialize a new run
wandb.init(project='easy-vqa-pytorch',
           name='VGG19-4FC-SBERT-Tanh-40Epoch-Med',
```

```
            notes='ninth run')


# WandB – Config is a variable that holds and saves hyperparameters and inputs
config = wandb.config          # Initialize config
config.batch_size = 32         # input batch size for training (default: 64)
config.test_batch_size = 32    # input batch size for testing (default: 1000)
config.epochs = 40             # number of epochs to train (default: 10)
config.lr = 0.01               # learning rate (default: 0.01)
config.momentum = 0.5          # SGD momentum (default: 0.5)
config.no_cuda = False         # disables CUDA training
config.log_interval = 10     # how many batches to wait before logging training status


if torch.cuda.is_available(): device = torch.device("cuda:0")
kwargs = {'num_workers': 1, 'pin_memory': True} if torch.cuda.is_available() else {}


# Now we load our training and test datasets initialize the train, validation, and test data loaders

train_dataset = CustomDataset(train_X_ims, train_X_seqs, train_Y)
test_dataset = CustomDataset(test_X_ims, test_X_seqs, test_Y)
trainloader = DataLoader(train_dataset, shuffle=True, batch_size=config.batch_size)
testloader = DataLoader(test_dataset, batch_size=config.test_batch_size)


# Initialize our model, recursively go over all modules and convert their parameters and buffers to CUDA tensors (if device i
model = VQA_v2(embedding_size = 384, num_answers = num_answers).to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=config.lr,
                      momentum=config.momentum )

# WandB – wandb.watch() automatically fetches all layer dimensions, gradients, model parameters and logs them automatically t
# Using log="all" log histograms of parameter values in addition to gradients
wandb.watch(model, log="all")
train_losses, valid_losses = [], []

for epoch in range(config.epochs):
    train_loss = train_loop(model, optimizer, criterion, trainloader)
    valid_loss = validate_loop(model, criterion, testloader)

    tqdm.write(
        f'epoch #{epoch + 1:3d}\ttrain_loss: {train_loss:.2e}\tvalid_loss: {valid_loss:.2e}\n',
    )

    train_losses.append(train_loss)
    valid_losses.append(valid_loss)

    wandb.log({
      "Epoch": epoch,
      "Training Loss": train_loss,
      "Validation Loss": valid_loss})
```

```
epoch # 13      train_loss: 9.67e-03    valid_loss: 9.92e-03
epoch # 14      train_loss: 9.55e-03    valid_loss: 9.68e-03
epoch # 15      train_loss: 9.44e-03    valid_loss: 9.35e-03
epoch # 16      train_loss: 9.31e-03    valid_loss: 9.33e-03
epoch # 17      train_loss: 9.18e-03    valid_loss: 9.10e-03
epoch # 18      train_loss: 9.21e-03    valid_loss: 9.09e-03
epoch # 19      train_loss: 8.96e-03    valid_loss: 9.01e-03
epoch # 20      train_loss: 9.10e-03    valid_loss: 8.77e-03
epoch # 21      train_loss: 8.77e-03    valid_loss: 8.80e-03
epoch # 22      train_loss: 8.65e-03    valid_loss: 8.56e-03
epoch # 23      train_loss: 8.55e-03    valid_loss: 8.61e-03
epoch # 24      train_loss: 8.48e-03    valid_loss: 8.49e-03
epoch # 25      train_loss: 8.42e-03    valid_loss: 8.30e-03
```

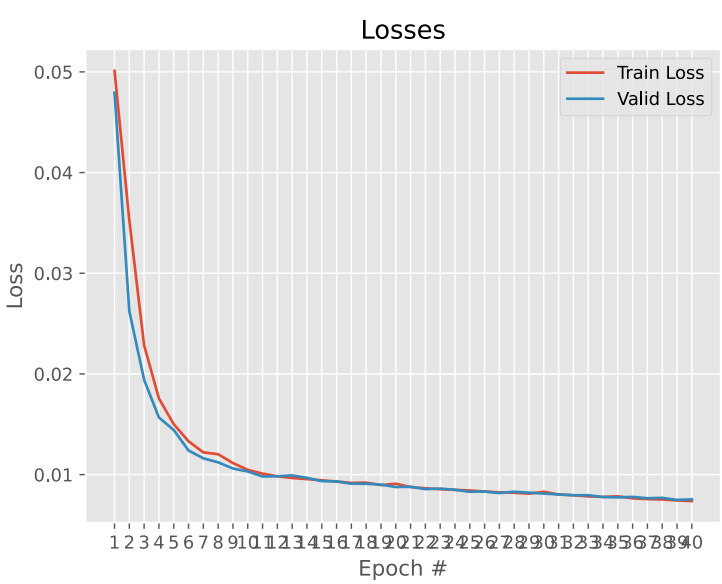
Loss charts

```
epoch # 29      train_loss: 8.11e-03    valid_loss: 8.21e-03
```

```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_formats = ['svg']
plt.style.use('ggplot')


epoch_ticks = range(1, epoch + 2)
plt.plot(epoch_ticks, train_losses)
plt.plot(epoch_ticks, valid_losses)
plt.legend(['Train Loss', 'Valid Loss'])
plt.title('Losses')
plt.xlabel('Epoch #')
plt.ylabel('Loss')
```

```
plt.xticks(epoch_ticks)
plt.show()
```



Validation Accuracy

```
model.eval()
model.to(device)
num_correct = 0
num_samples = 0
predictions = []
answers = []

with torch.no_grad():
    for image, text, label in testloader:
        image, text, label =  image.to(device), text.to(device), label.to(device)
        probs = model.forward(image, text)

        _, prediction = probs.max(1)
        predictions.append(prediction)

        answer = torch.argmax(label, dim=1)
        answers.append(answer)

        num_correct += (prediction == answer).sum()
        num_samples += prediction.size(0)

    valid_acc = (f'Got {num_correct} / {num_samples} with accuracy {float(num_correct)/float(num_samples)*100:.2f}')
    print(valid_acc)

    wandb.log({
    "Validation Accuracy": round(float(num_correct)/float(num_samples)*100, 2)})
```

```
 Got 8830 / 9673 with accuracy 91.29
```

Saving and Loading the model

```
import os
from pathlib import Path

project_path = '/content/drive/MyDrive/FSDL'  # Make sure this is your intended path
model_path = os.path.join(project_path, 'VGG19-Sbert-40')

model = VQA_v2(embedding_size=384, num_answers=13)
model.load_state_dict(torch.load(model_path, map_location='cuda:0'))  # or 'cpu' if you're not using CUDA
model.eval()
```

```
VQA_v2(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
          (3): ReLU(inplace=True)
          (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (6): ReLU(inplace=True)
          (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (8): ReLU(inplace=True)
          (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (11): ReLU(inplace=True)
          (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (13): ReLU(inplace=True)
          (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (15): ReLU(inplace=True)
          (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (17): ReLU(inplace=True)
          (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (20): ReLU(inplace=True)
          (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (22): ReLU(inplace=True)
          (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (24): ReLU(inplace=True)
          (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (26): ReLU(inplace=True)
          (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
          (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (29): ReLU(inplace=True)
          (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (31): ReLU(inplace=True)
          (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (33): ReLU(inplace=True)
          (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (35): ReLU(inplace=True)
          (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        )
        (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
        (classifier): Sequential(
          (0): Linear(in_features=25088, out_features=512, bias=True)
          (1): Tanh()
          (2): Linear(in_features=512, out_features=128, bias=True)
          (3): Tanh()
          (4): Linear(in_features=128, out_features=32, bias=True)
        )
        (fc2): Linear(in_features=384, out_features=64, bias=True)
        (fc3): Linear(in_features=64, out_features=32, bias=True)
        (fc4): Linear(in_features=64, out_features=32, bias=True)
        (fc5): Linear(in_features=32, out_features=13, bias=True)
      )
```

```
import torch
from urllib.request import urlopen
from PIL import Image
import torchvision.transforms as transforms

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

def load_and_process_image_url(url):
    # Loads image from URL and converts to Tensor
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    im = Image.open(urlopen(url)).convert('RGB')
    im = transform(im)
    return im

url = "https://www.nicepng.com/png/detail/16-163438_circle-clipart-sky-blue-clip-art-blue-circle.png"
image = load_and_process_image_url(url)
image = image.unsqueeze(0).to(device)  # Add batch dimension and send to device

text = 'What shape is this?'
text = st_model.encode(text)
text = torch.tensor(text, dtype=torch.float).unsqueeze(0).to(device)  # Add batch dimension and send to device

model = VQA_v2(embedding_size = 384, num_answers = 13).to(device)
model.eval()

with torch.no_grad():  # Ensure no gradients are computed for this forward pass
    probs = model(image, text)
    answer_idx = torch.argmax(probs, dim=1)  # Get index of answer with highest probability
    answer_text = [all_answers[idx] for idx in answer_idx.cpu().numpy()]  # Convert index to answer text and ensure it's on C
    print(answer_text)
```