Install easy-vqa library

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

```
Collecting easy-vqa
  Downloading easy_vqa-1.0-py3-none-any.whl (3.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 2.2 MB/s eta 0:00:00
Installing collected packages: easy-vqa
Successfully installed easy-vqa-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-2-c6865f93d0ef>:33: FutureWarning: The input object of type 'Tensor' is an array-like implementing one of the
  train_X_ims = np.array([train_ims[id] for id in train_image_ids])
<ipython-input-2-c6865f93d0ef>:33: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a l
  train_X_ims = np.array([train_ims[id] for id in train_image_ids])
<ipython-input-2-c6865f93d0ef>:34: FutureWarning: The input object of type 'Tensor' is an array-like implementing one of the
  test_X_ims = np.array([test_ims[id] for id in test_image_ids])
<ipython-input-2-c6865f93d0ef>:34: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a l
  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}')
```

```
Collecting sentence-transformers
  Downloading sentence-transformers-2.2.2.tar.gz (85 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.0/86.0 kB 3.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting transformers<5.0.0,>=4.6.0 (from sentence-transformers)
  Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 66.7 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (4.66.1)
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (2.1.0+c
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (0.16.0+c
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (1.23.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (1.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (1.11.3)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (from sentence-transformers) (3.8.1)
Collecting sentencepiece (from sentence-transformers)
  Downloading sentencepiece-0.1.99-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 86.1 MB/s eta 0:00:00
Collecting huggingface-hub>=0.4.0 (from sentence-transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 39.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.4.0->sentence-tr
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.4.0->sen
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.4.0->sentence-tr
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.4.0->sentence
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.4.0->sent
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.6.0->sentence-transformers) (
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.6.0->sentence-transformers
```

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 lay
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., 0.])
data type <class 'torch.Tensor'>
```

Downloading (…)_Pooling/config.json: 100% 190/190 [00:00<00:00, 13.0kB/s]

The Model

```
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)
    resnet = torchvision.models.resnet50(pretrained=True)
    num_ftrs = resnet.fc.in_features
    resnet.fc = torch.nn.Sequential(
        torch.nn.Linear(num_ftrs, 512),
        torch.nn.Tanh(),
        torch.nn.Linear(512, 128),
        torch.nn.Tanh(),
        torch.nn.Linear(128, 32)
    )
```

```
    self.mdl = resnet

    # 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.mdl(x)                             # (batch_size, 32)

    # 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)

          # 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
```

```
    Requirement already satisfied: WandB in /usr/local/lib/python3.10/dist-packages (0.15.12)
    Requirement already satisfied: Click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-packages (from WandB) (8.1.7)
    Requirement already satisfied: GitPython!=3.1.29,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (3.1.40)
    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)
    Requirement already satisfied: sentry-sdk>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (1.32.0)
    Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from WandB) (0.4.0)
    Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from WandB) (6.0.1)
    Requirement already satisfied: pathtools in /usr/local/lib/python3.10/dist-packages (from WandB) (0.1.2)
    Requirement already satisfied: setproctitle in /usr/local/lib/python3.10/dist-packages (from WandB) (1.3.3)
    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) (3.20.3)
    Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from docker-pycreds>=0.4.0->WandB) (1.1
    Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from GitPython!=3.1.29,>=1.0.0->Wa
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->WandB) (3.4)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->WandB)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->WandB)
    Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1->GitPython!=3
    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_remount=Tr
```

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='ResNet-4FC-SBERT-Tanh-20Epoch-Med',
```

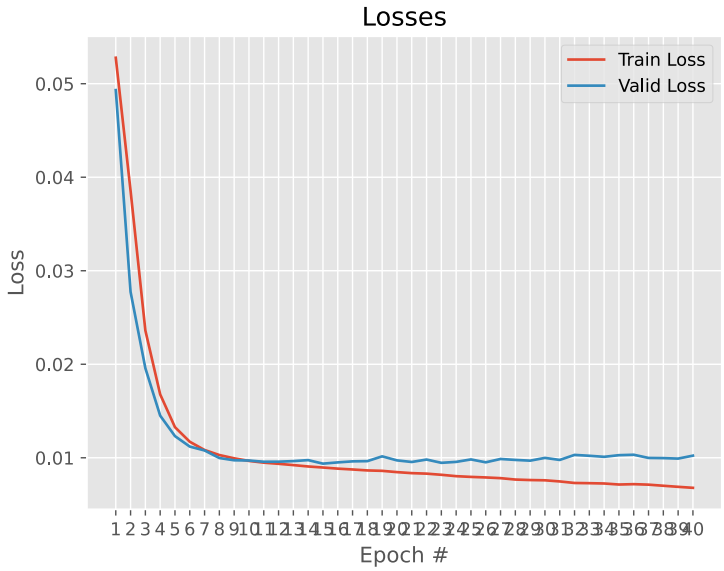
```
            notes='ninth run')


    # 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 is se
    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 to yo
    # 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})
```

```
wandb: Currently logged in as: yuqingsu9. Use `wandb login --relogin` to force r
Tracking run with wandb version 0.15.12
Run data is saved locally in /content/wandb/run-20231029_215333-lab84zkc
Syncing run ResNet-4FC-SBERT-Tanh-20Epoch-Med to Weights & Biases (docs)
View project at https://wandb.ai/yuqingsu9/easy-vqa-pytorch
View run at https://wandb.ai/yuqingsu9/easy-vqa-pytorch/runs/lab84zkc
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWa
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /roo
100%|██████████| 97.8M/97.8M [00:01<00:00, 93.9MB/s]
epoch #  1      train_loss: 5.28e-02    valid_loss: 4.93e-02

epoch #  2      train_loss: 3.86e-02    valid_loss: 2.77e-02

epoch #  3      train_loss: 2.36e-02    valid_loss: 1.96e-02

epoch #  4      train_loss: 1.68e-02    valid_loss: 1.45e-02

epoch #  5      train_loss: 1.33e-02    valid_loss: 1.23e-02

epoch #  6      train_loss: 1.17e-02    valid_loss: 1.12e-02

epoch #  7      train_loss: 1.08e-02    valid_loss: 1.08e-02

epoch #  8      train_loss: 1.03e-02    valid_loss: 9.97e-03

epoch #  9      train_loss: 9.94e-03    valid_loss: 9.73e-03

epoch # 10      train_loss: 9.66e-03    valid_loss: 9.71e-03

epoch # 11      train_loss: 9.48e-03    valid_loss: 9.58e-03

epoch # 12      train_loss: 9.36e-03    valid_loss: 9.58e-03

epoch # 13      train_loss: 9.22e-03    valid_loss: 9.64e-03

epoch # 14      train_loss: 9.07e-03    valid_loss: 9.75e-03

epoch # 15      train_loss: 8.96e-03    valid_loss: 9.37e-03

epoch # 16      train_loss: 8.84e-03    valid_loss: 9.51e-03

epoch # 17      train_loss: 8.75e-03    valid_loss: 9.62e-03

epoch # 18      train_loss: 8.65e-03    valid_loss: 9.64e-03

epoch # 19      train_loss: 8.60e-03    valid_loss: 1.02e-02

epoch # 20      train_loss: 8.47e-03    valid_loss: 9.72e-03

epoch # 21      train_loss: 8.36e-03    valid_loss: 9.55e-03

epoch # 22      train_loss: 8.30e-03    valid_loss: 9.81e-03

epoch # 23      train_loss: 8.18e-03    valid_loss: 9.46e-03
```

Loss charts

```
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 8566 / 9673 with accuracy 88.56
```

Saving and Loading the model

```
torch.save(model.state_dict(), project_path/'Resnet-Sbert-40')
```

```
model = VQA_v2(embedding_size = 384, num_answers = 13)
model.load_state_dict(torch.load(project_path/'Resnet-Sbert-40')) # Im loading my best model
model.eval()
```

```
            (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)
          )
        )
        (layer4): Sequential(
          (0): Bottleneck(
            (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
            (downsample): Sequential(
              (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
              (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
          )
          (1): Bottleneck(
            (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
          )
          (2): Bottleneck(
            (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
          )
        )
        (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
        (fc): Sequential(
          (0): Linear(in_features=2048, 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)
    )
```

```
from urllib.request import urlopen
from PIL import Image

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

text = 'What shape is this?'
text = st_model.encode(text)
text = torch.tensor(text, dtype=torch.float)
text = text.unsqueeze(0)

probs = model.forward(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] # convert index to answer text
print(answer_text)
```

```
['triangle']
```