Deep Learning Image Captioning Project Documentation

Folder Structure

```
├──{extracted folder name}
│     ├──datasets
│     │     ├──annotations
│     │     ├──test2014
│     │     ├──train2014
│     │     └──val2014
│     ├──utils
│     │     ├──models.py
│     │     └──data_loader.py
│     ├──scratch
│     │     ├──models
│     │           ├──final_coop-generator-3-25000.pkl
│     │           └──final_coop-discriminator-3-25000.pkl
│     ├──main.py
│     ├──GUI.py
│     ├──vocabulary.py
│     ├──vocab.pkl
│     ├──requirements.txt
│     ├──VideoDL.mp4
│     ├──vocab.pkl
│     ├──Notebook main latest training with loss progress.ipynb
│     ├──Notebook main latest training with testing.ipynb
```

Installation

Python version: 3.6
Torch version: 1.2.0
Trained on the SUTD GPU Cluster
`!pip install pycocotools`
`!pip install git+https://github.com/salaniz/pycocoevalcap`
`!pip install > requirements.txt`

```
import nltk
nltk.download('punkt')
```

Execution

Training

To retrain the model:

- Linux Users can execute the following:

  `python3 main.py`

- Windows Users can execute the following:

```
python main.py
```

To execute the GUI:

- Linux Users can execute the following:

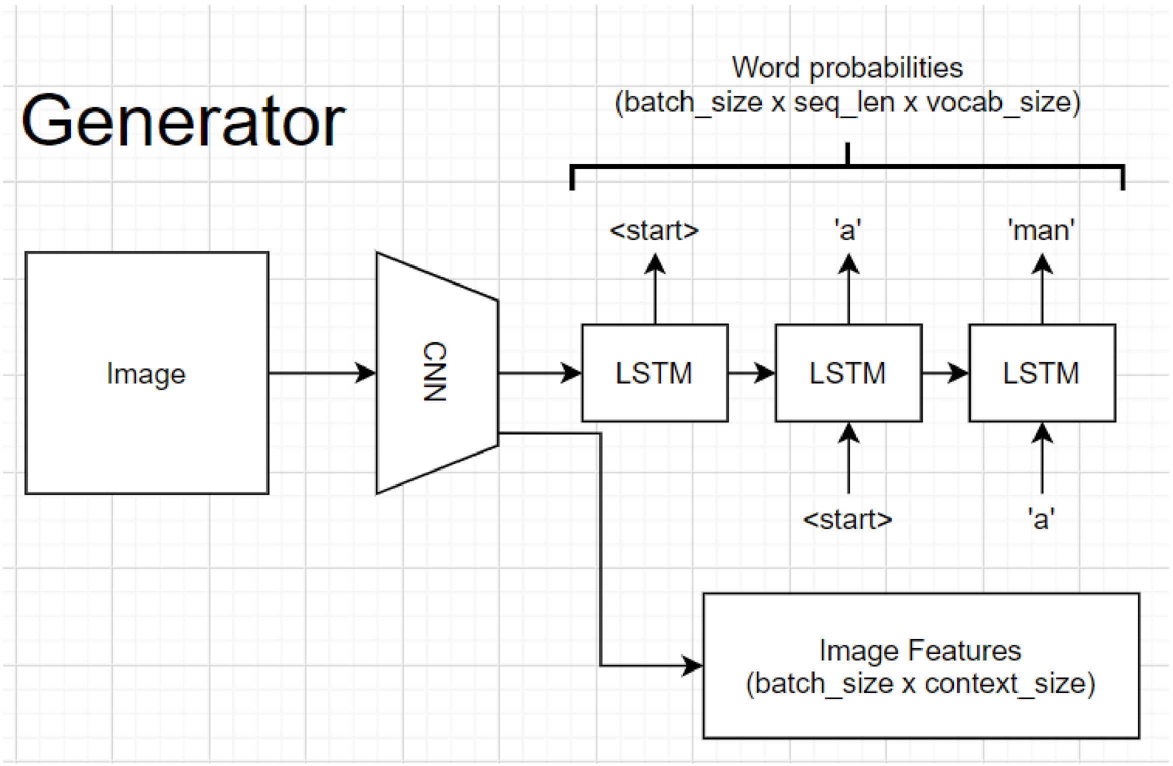
```
python3 GUI.py
```

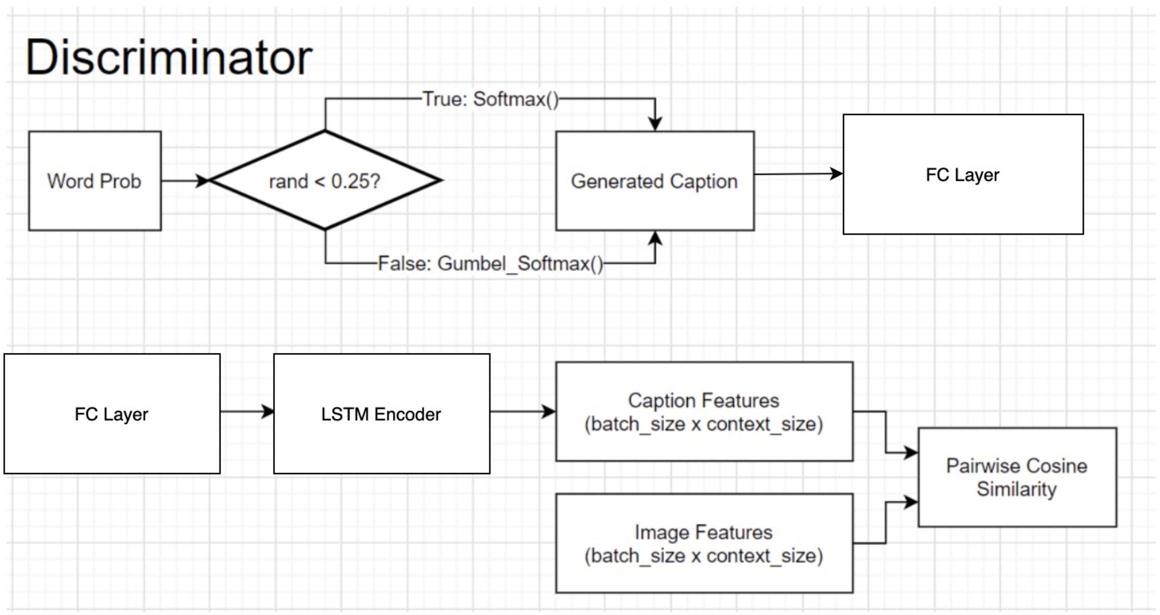
- Windows Users can execute the following:

```
python GUI.py
```

Model Design

Cooperative Image Captioning





Motivation:
Not only do we want to create realistic captions, but also to create captions unique to the input image.

Hence, we needed to introduce a form of discriminative loss to the provided tutorial image captioning model.

Architecture

The architecture is heavily inspired by the following paper:

Gilad Vered, Gal Oren, Yuval Atzmon, Gal Chechik; The IEEE International Conference on Computer Vision (ICCV), 2019, pp. 8898-8907

Training Hyperparameters

Batch size = 256
Epochs = 3
Embedding size = 300
Context size = 300
Hidden size = 512

Image Pre-Processing for training

```
transform_train = transforms.Compose([
    transforms.Resize(256),                          # smaller edge of image resized
    transforms.RandomCrop(224),                      # get 224x224 crop from random l
    transforms.RandomHorizontalFlip(),               # horizontally flip image with p
    transforms.ToTensor(),                           # convert the PIL Image to a ten
    transforms.Normalize((0.485, 0.456, 0.406),      # normalize image for pre-traine
                         (0.229, 0.224, 0.225))])
```

Training set is pre-processed firstly by resizing each image to 256, doing random crops of size 224, and doing random horizontal flip on the images. Once the images are pre-processed, they are then converted into tensors and normalised using the following values:

- mean: (0.485, 0.456, 0.406)
- std : (0.229, 0.224, 0.225)

Vocabulary Preprocessing

Vocabulary preprocessing is heavily inspired by the Pytorch Tutorial with some changes specific to the dataset.

Yunjey Choi Pytorch Image Captioning Tutorial

Models

EncoderCNN Model

```python
class EncoderCNN(nn.Module):
    def __init__(self, embed_size):
        super(EncoderCNN, self).__init__()
        resnet = models.resnet50(pretrained=True)
        for param in resnet.parameters():
            param.requires_grad_(False)

        modules = list(resnet.children())[:-1]
        self.resnet = nn.Sequential(*modules)
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)
        self.train_params = list(self.embed.parameters())

    def forward(self, images):
        features = self.resnet(images)
        features = features.view(features.size(0), -1)
        features = self.embed(features)
        return features
```

Get image features using a resnet50 backbone similar to how it is done in the Pytorch Image Captioning Tutorial. The EncoderCNN takes in an image, gets the features of the image and embed the features using a fully-connected layer for the decoderRNN model. This model is part of the Generator model explained in the following sections.

DecoderRNN Model

```
class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=2):
        super().__init__()
        self.embedding_layer = nn.Embedding(vocab_size, embed_size)

        self.lstm = nn.LSTM(input_size = embed_size,hidden_size = hidden_size,
                            num_layers = num_layers, batch_first = True)

        self.linear = nn.Linear(hidden_size, vocab_size)
        self.train_params = list(self.parameters())

    def forward(self, features, captions):
        captions = captions[:, :-1]
        embed = self.embedding_layer(captions)
        embed = torch.cat((features.unsqueeze(1), embed), dim = 1)
        lstm_outputs, _ = self.lstm(embed)
        out = self.linear(lstm_outputs)

        return out

    def sample(self, inputs, states=None, max_len=20):
        " accepts pre-processed image tensor (inputs) and returns predicted sentence
        output_sentence = []
        for i in range(max_len):
            lstm_outputs, states = self.lstm(inputs, states)
            lstm_outputs = lstm_outputs.squeeze(1)
            out = self.linear(lstm_outputs)
            last_pick = out.max(1)[1]
            output_sentence.append(last_pick.item())
            inputs = self.embedding_layer(last_pick).unsqueeze(1)

        return output_sentence

    def beam_sample(self, inputs, states=None, max_len=20, k=1):
        " accepts pre-processed image tensor (inputs) and returns predicted sentence
        possible_seq = [(1, inputs, states)]
        for i in range(max_len):
            to_pick = []
            for probs,seq,states in possible_seq:
                inputs = self.embedding_layer(seq[-1])
                lstm_outputs, states = self.lstm(inputs, states)
                out = self.linear(lstm_outputs).squeeze(0)
                sorted_out, indices = torch.sort(out, 1)

                for j in range(k):
                    to_pick.append((probs + nn.functional.log_softmax(sorted_out[i]),

            to_pick.sort(reverse=True)
            possible_seq = to_pick[:k]

        return to_pick[0]
```

DecoderRNN is part of the generator model, explained in the following sections, that takes in the features from the encoder and processes it for the caption generation. This model is designed using an LSTM layer to associate the features of image with the embedded captions. This model outputs unnormalised probability of a batch of captions. The unnormalised probability will then be used for further processing as part of the generator model.

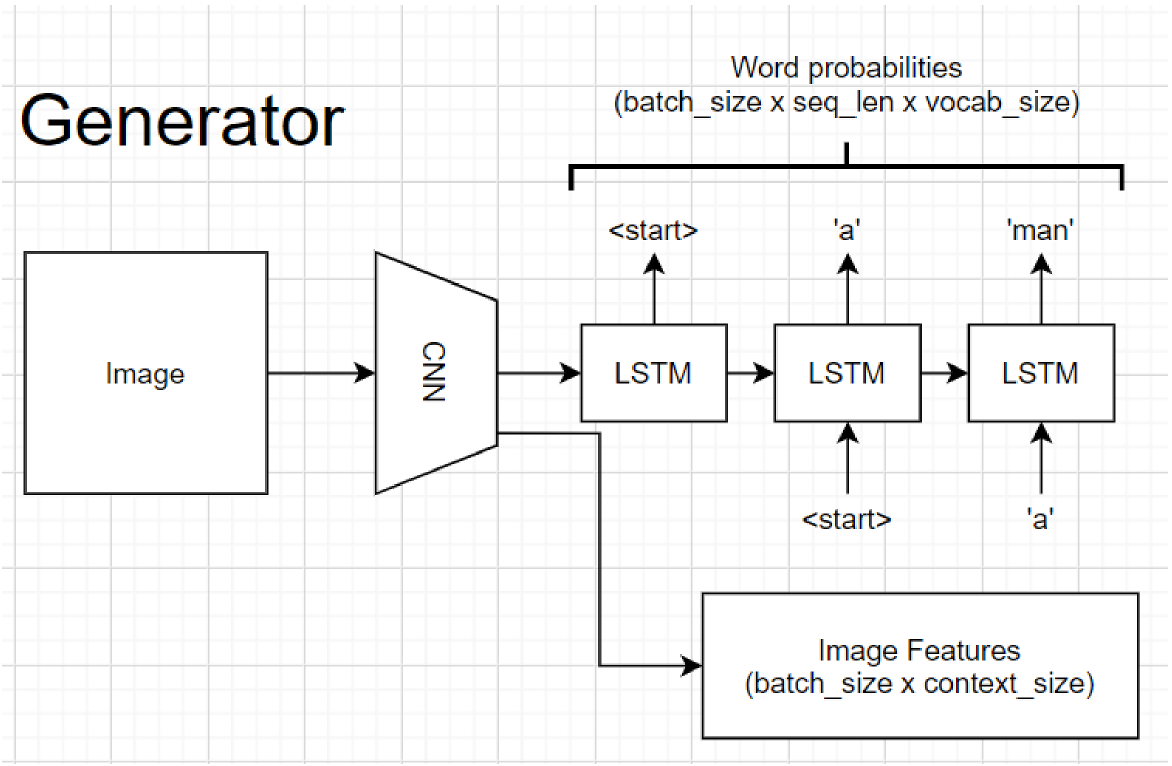
EncoderRNN Model

```
class EncoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, context_size, num_layers=
        super().__init__()
        self.lstm = nn.LSTM(input_size = embed_size,hidden_size = hidden_size,
                            num_layers = num_layers, batch_first = True)
        self.linear = nn.Linear(hidden_size, context_size)
        self.train_params = list(self.parameters())

    def forward(self, captions):
        lstm_outputs, _ = self.lstm(captions)
        out = self.linear(lstm_outputs[:,-1,:].squeeze(1))
        return out
```

EncoderRNN is part of the discriminator model which is used to compute the caption feature vectors to calculate the similarity between the generated captions and the ground truth captions. This model takes in the generated captions from the generator model and outputs the caption features vectors to be used for the training.

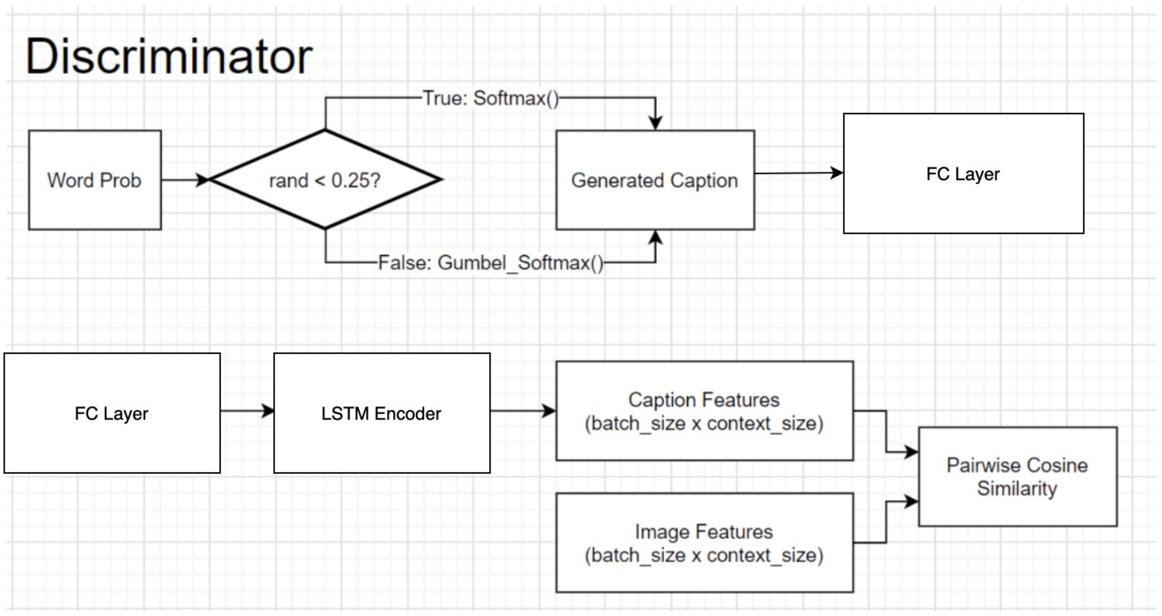
Generator Model



Generator takes in the `Extracted Features` and `Caption Features` from EncorderCNN and DecoderRNN(LSTM) respectively.

It outputs a generated probability of caption word and the extracted image features. These values will then be passed to and trained with the Discriminator.

Discriminator Model



The FC layer acts like an Embedding layer, but we needed to allow the model to take in non-discrete inputs.

The method of sampling for the discriminator follow the Cooperative Image Captioning model, which uses Partial-Sampling-Straight-Through (PSST). Motivation for this approach is to avoid the main issues regarding sampling for text generation.

Sampling for text generation usually involves using argmax, which is not differentiable. This can be avoided by using the Gumbel Softmax function which samples in a differentiable manner. However, Gumbel Softmax has high variance and bias, as described in the Cooperative Image Captioning paper. So instead, the paper proposes PSST, which passes the full continuous values from the Generator to the Discriminator with probability rho (rho = 0.25), and passes a 1-hot representation using Gumbel Softmax with probability 1-rho.

Next, the generated caption (FC Layer) passes through the EncoderRNN, which computes the caption features vector. We then use compute the pairwise cosine similarity between the image features vectors and caption features vectors which we pass to the loss function to minimise.

Loss Function

```
        max_loss_1 = 0
        max_loss_2 = 0
        pos_pair_term = cosineSim(cap_feats[0], img_feats[0])
        for i in range(1,batch_size):
            max_loss_1 += max(0, 1 - pos_pair_term + cosineSim(cap_feats[0], img_feat
            max_loss_2 += max(0, 1 - pos_pair_term + cosineSim(cap_feats[i], img_feat



        # Calculate the batch loss.
        loss_A = criterion_A(out.view(-1, vocab_size), captions.view(-1))
        loss_B = (max_loss_1 + max_loss_2)/(batch_size-1)

        B_weight = 0.5
        loss = loss_A + B_weight*loss_B
```

Following a similar approach to the Cooperative Image Captioning model, our loss is based on 2 components: Natural Loss and Discriminability Loss.

For the Natural Loss, we used the CrossEntropyLoss method to compute the perplexity of the model given the ground truth caption, which is a measure of how well a probability model predicts a sample. This loss is used to ensure the language of the captions are realistic.

For the Discriminability Loss, we aim to have generated captions unique and specific to the image. To do this, we compute the similarity of a candidate caption and its source image and similarity of that candidate caption and distractor images. We aim to maximise the similarity of the generated caption and the source image, while minimising the similarity with the distractor images.

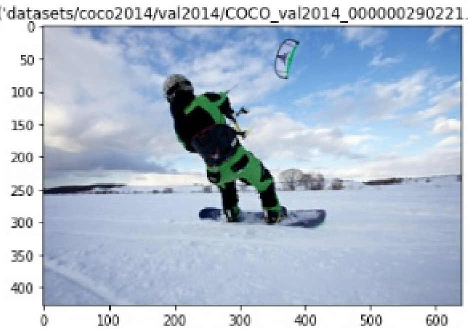
This would increase the model's ability to perform in self-retrieval, where given a set of images (distractor images + source image) and the generated caption, one can identify the source image from the set of distractor images. A similar logic is applied to identify the correct caption given the source image and a set of captions (distractor captions + generated caption).

Performance

The model is evaluated using the validation set (`val2014`).

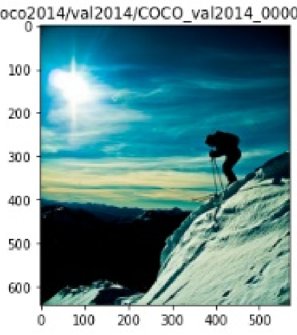
The following are the sample images generated.

```
('datasets/coco2014/val2014/COCO_val2014_000000290221.jpg',)
```



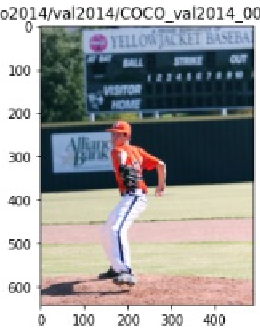
```
A man riding a snowboard down a snow covered slope . on a snow
```

```
('datasets/coco2014/val2014/COCO_val2014_000000111526.jpg',)
```



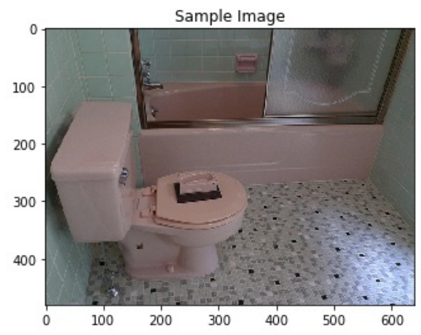
```
A person on a snowboard in the snow . on a snowy
```

```
('datasets/coco2014/val2014/COCO_val2014_000000365701.jpg',)
```



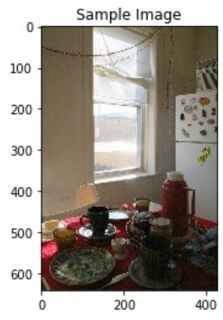
```
A baseball player swinging a bat at a ball . on the other
```

Using Bleu-1, Bleu-2, Bleu-3, Bleu-4, and ROUGE-L evaluation methods,
the following are some samples of the prediction scores:



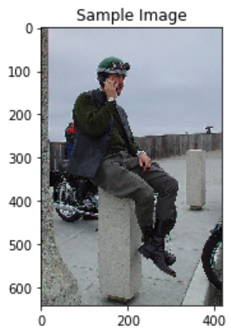
```
The ground truth caption of the image tensor([63882]) : ['The bathroom has a pink tub and pink toilet.']
The generatred caption of the image tensor([63882]) : ['A bathroom with a toilet and a sink in it on a toilet .']
{'testlen': 14, 'reflen': 9, 'guess': [14, 13, 12, 11], 'correct': [3, 0, 0, 0]}
ratio: 1.5555555553827163
```

```
{'Bleu_1': 0.2142857142704082,
 'Bleu_2': 4.059989714404599e-09,
 'Bleu_3': 1.111619628767246e-11,
 'Bleu_4': 5.944549994012258e-13,
 'ROUGE_L': 0.271513353115727}
```



```
{'testlen': 13, 'reflen': 11, 'guess': [13, 12, 11, 10], 'correct': [6, 1, 0, 0]}
ratio: 1.181818181710744
The ground truth caption of the image tensor([345245]) : ['A tablein a kichen with dishes and a canister on it.']
The generatred caption of the image tensor([345245]) : ['A kitchen with a stove and a stove top on a counter  . ']
{'testlen': 13, 'reflen': 11, 'guess': [13, 12, 11, 10], 'correct': [6, 1, 0, 0]}
ratio: 1.181818181710744
```

```
{'Bleu_1': 0.46153846150295863,
 'Bleu_2': 0.1961161351224697,
 'Bleu_3': 1.5177887247853536e-06,
 'Bleu_4': 4.324227075083586e-09,
 'ROUGE_L': 0.39559014267185466}
```



```
{'testlen': 11, 'reflen': 10, 'guess': [11, 10, 9, 8], 'correct': [2, 0, 0, 0]}
ratio: 1.09999999989
The ground truth caption of the image tensor([363272]) : ['A man sitting on a pillar talking on the phone']
The generatred caption of the image tensor([363272]) : ['A woman is walking down a street with a umbrella  . ']
{'testlen': 11, 'reflen': 10, 'guess': [11, 10, 9, 8], 'correct': [2, 0, 0, 0]}
ratio: 1.09999999989
```

```
{'Bleu_1': 0.18181818180165296,
 'Bleu_2': 4.26401432670519e-09,
 'Bleu_3': 1.2641490044408468e-11,
 'Bleu_4': 7.088856801504128e-13,
 'ROUGE_L': 0.1781021897810219}
```

Sample Captions:

```
1. ('a bench on a hill overlooking a body of water',)
    A person riding a surfboard on a beach .
2. ('A street is empty and the traffic light is red.',)
    A street sign with a traffic light in the background.
3. ('A red cross sitting over a white toilet.',)
    A bathroom with a toilet and a sink in it on a white.
```

Average scores:

```
bleu1 = 0.32000085354820995
bleu2 = 0.1555914132856976
bleu3 = 0.04849434560587178
bleu4 = 0.025579018256153007
rougel =  0.3220487108487605
```

GUI

After training the models, we implemented a GUI based off python’s tkinter library.

```
from tkinter import *
from PIL import Image, ImageTk

if __name__ == '__main__':
    root = Toplevel()
    app = Window(root)
```

Dynamic Image Resizing has been implemented.

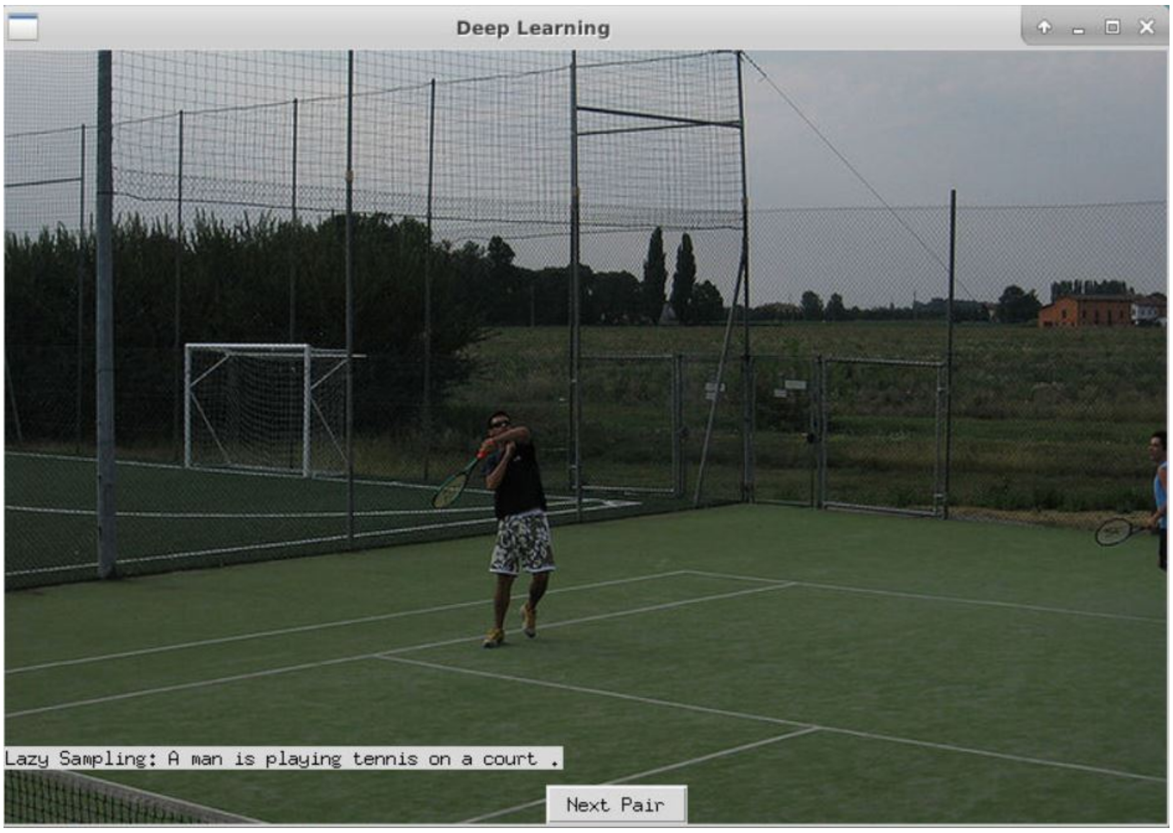
```
    def resizeimg(self, img):
        basewidth = 500
        wpercent = (basewidth/float(img.size[0]))
        hsize = int((float(img.size[1])*float(wpercent)))
        img = img.resize((basewidth,hsize), Image.ANTIALIAS)
        return img
```

The GUI calls the Generator, data_loader modules.

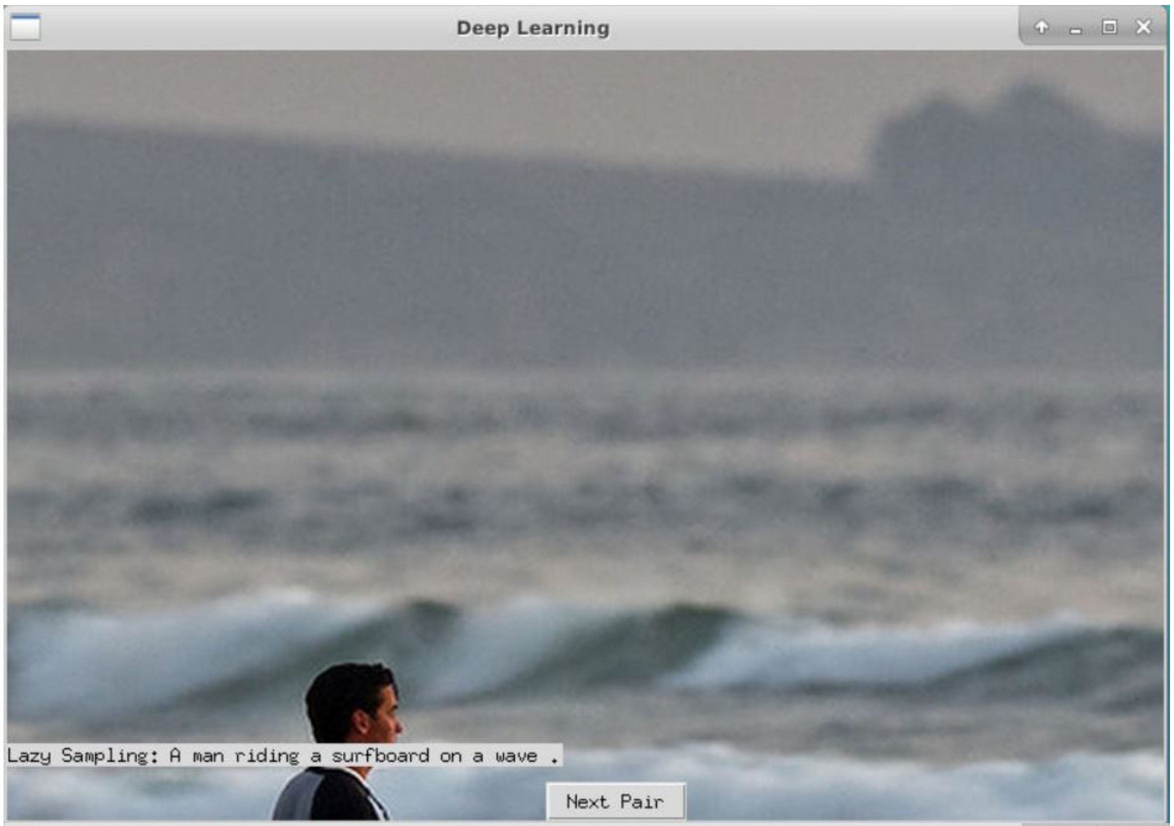
```
    def initmodels(self):
        transform_train = transforms.Compose([
        transforms.Resize(256),                          # smaller edge of image resi
        transforms.RandomCrop(224),                      # get 224x224 crop from rand
        transforms.RandomHorizontalFlip(),               # horizontally flip image wi
        transforms.ToTensor(),                           # convert the PIL Image to a
        transforms.Normalize((0.485, 0.456, 0.406),      # normalize image for pre-tr
                             (0.229, 0.224, 0.225))])
        self.testdata_loader = get_loader(transform=transform_train,
                             mode='test',
                             batch_size=1,
                             vocab_threshold=5,
                             vocab_from_file=True)

        embed_size = 300           # dimensionality of image and word embeddings
        hidden_size = 512          # number of features in hidden state of the RNN de
        vocab_size = len(self.testdata_loader.dataset.vocab)
        self.Generator = Generator(embed_size, hidden_size, vocab_size, embed_size)
        self.Generator.load_state_dict(torch.load('./models/decoder-1-26500.pkl'))
        self.encoder = self.Generator.cnn
        self.decoder = self.Generator.rnn
        self.encoder.to(device)
        self.decoder.to(device)
        self.encoder.eval()
        self.decoder.eval()
```

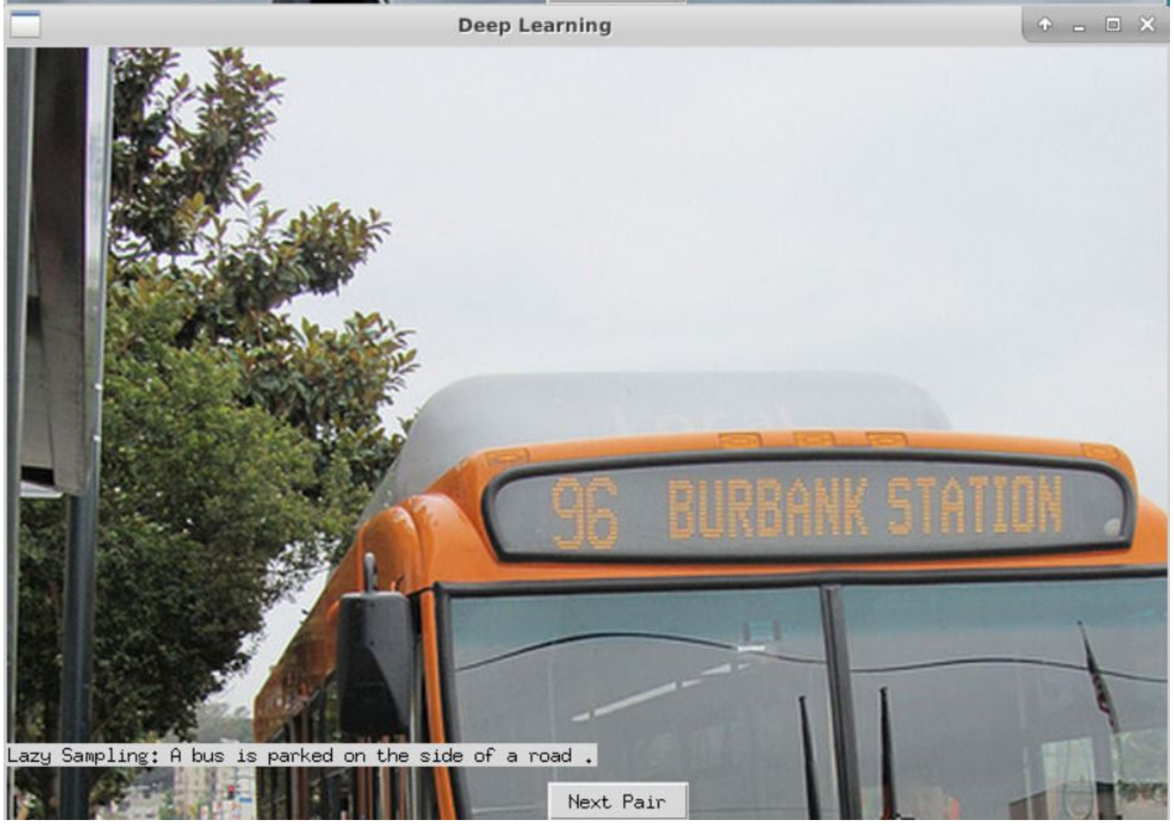
Some good predictions:



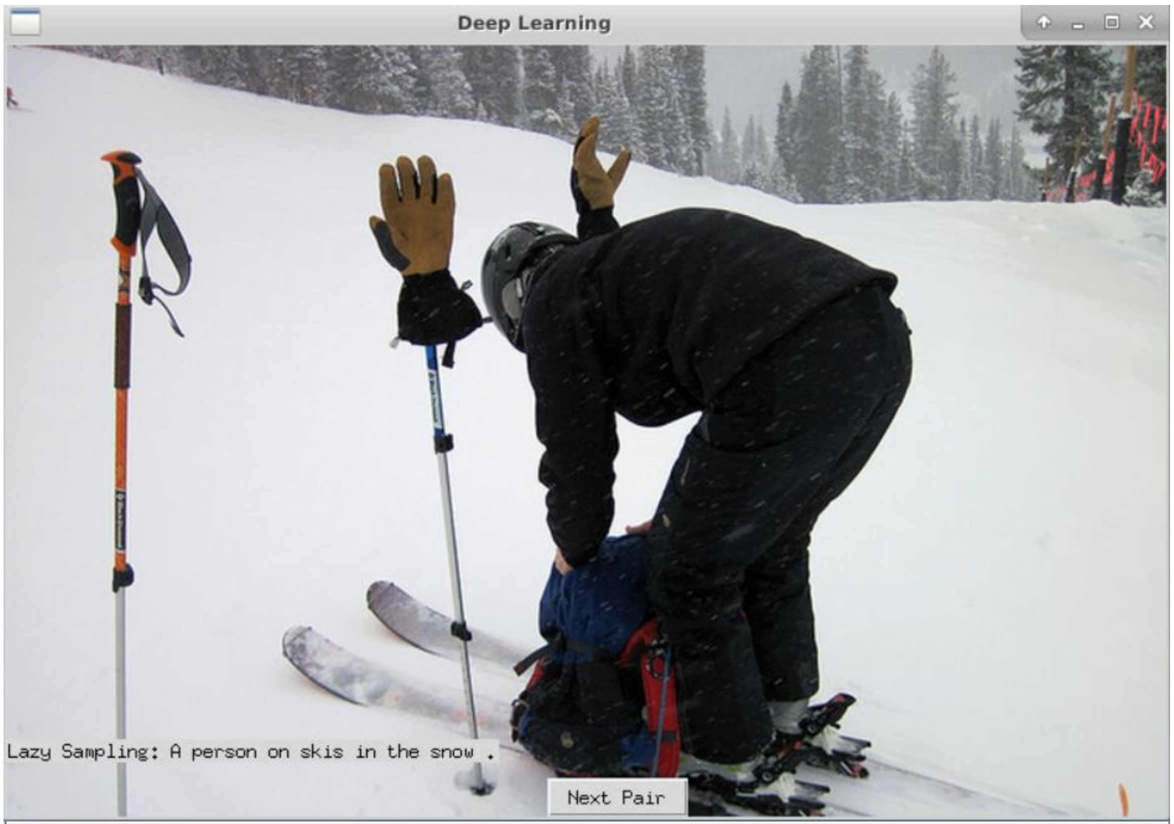
Lazy Sampling: A man is playing tennis on a court .



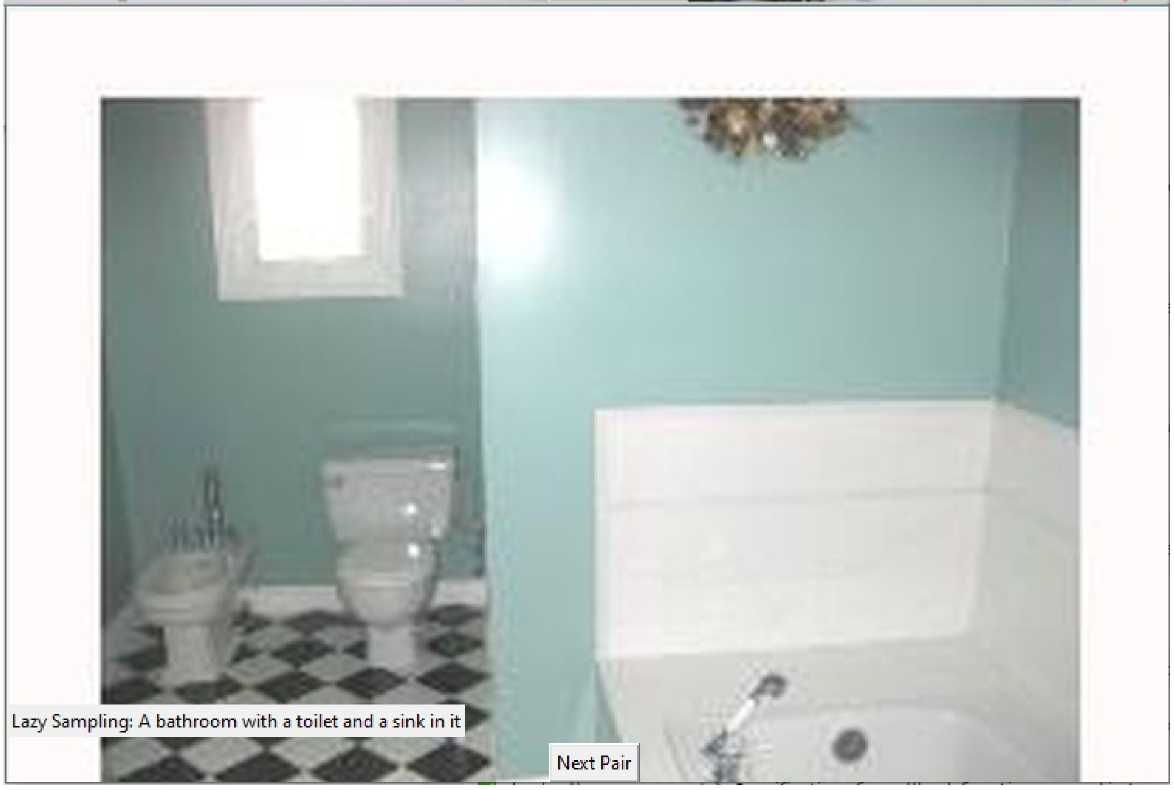
Lazy Sampling: A man riding a surfboard on a wave .



Lazy Sampling: A bus is parked on the side of a road .

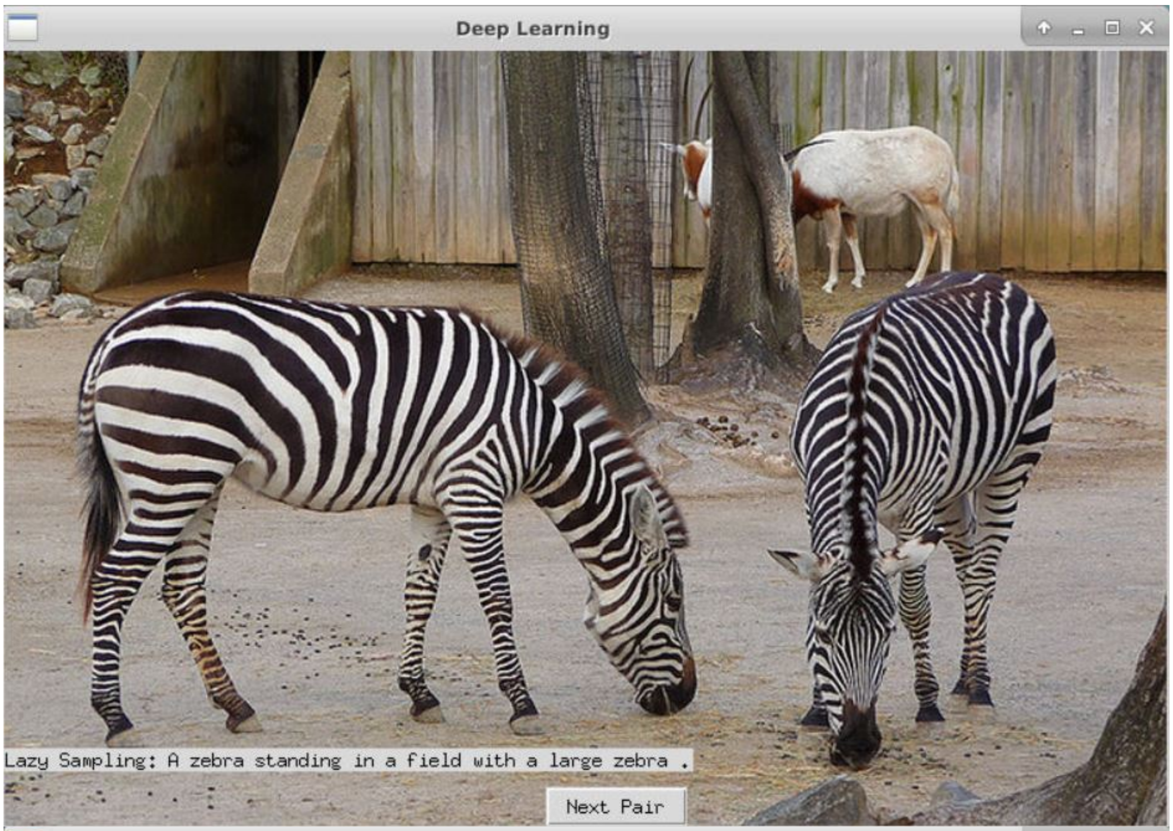


Lazy Sampling: A person on skis in the snow .

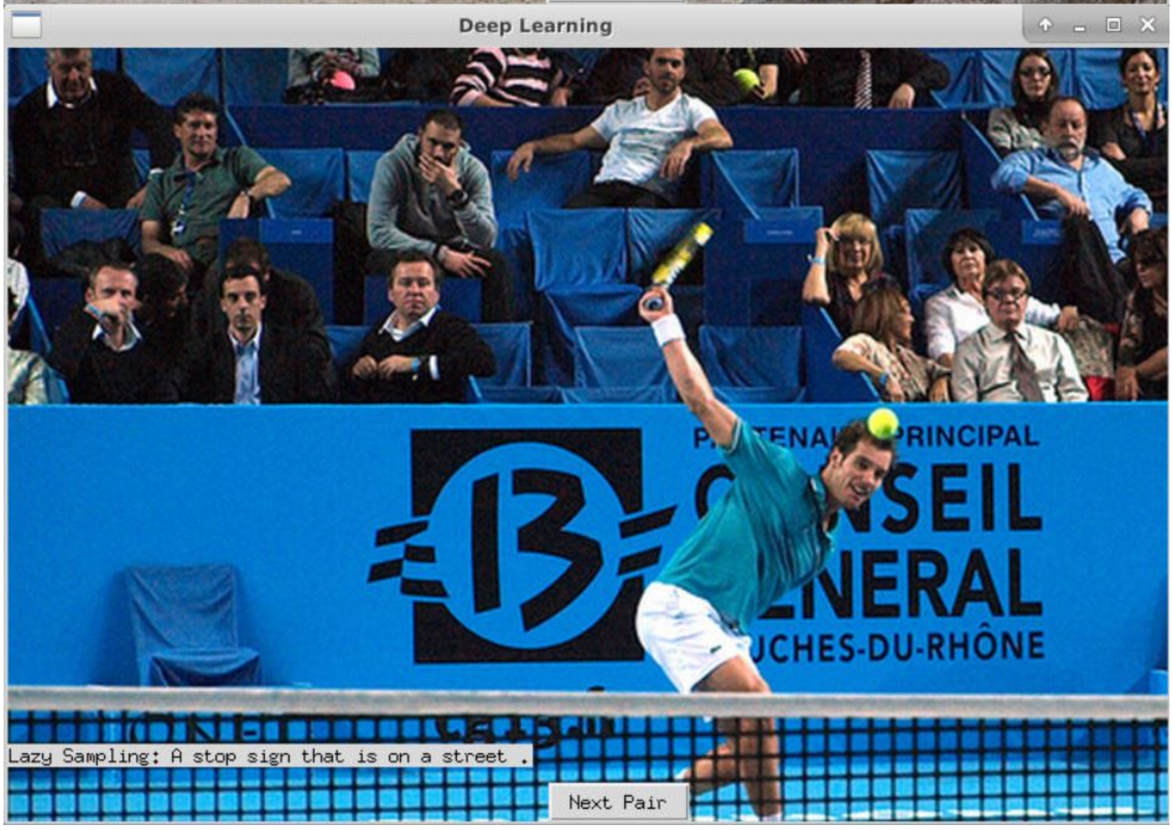


Lazy Sampling: A bathroom with a toilet and a sink in it

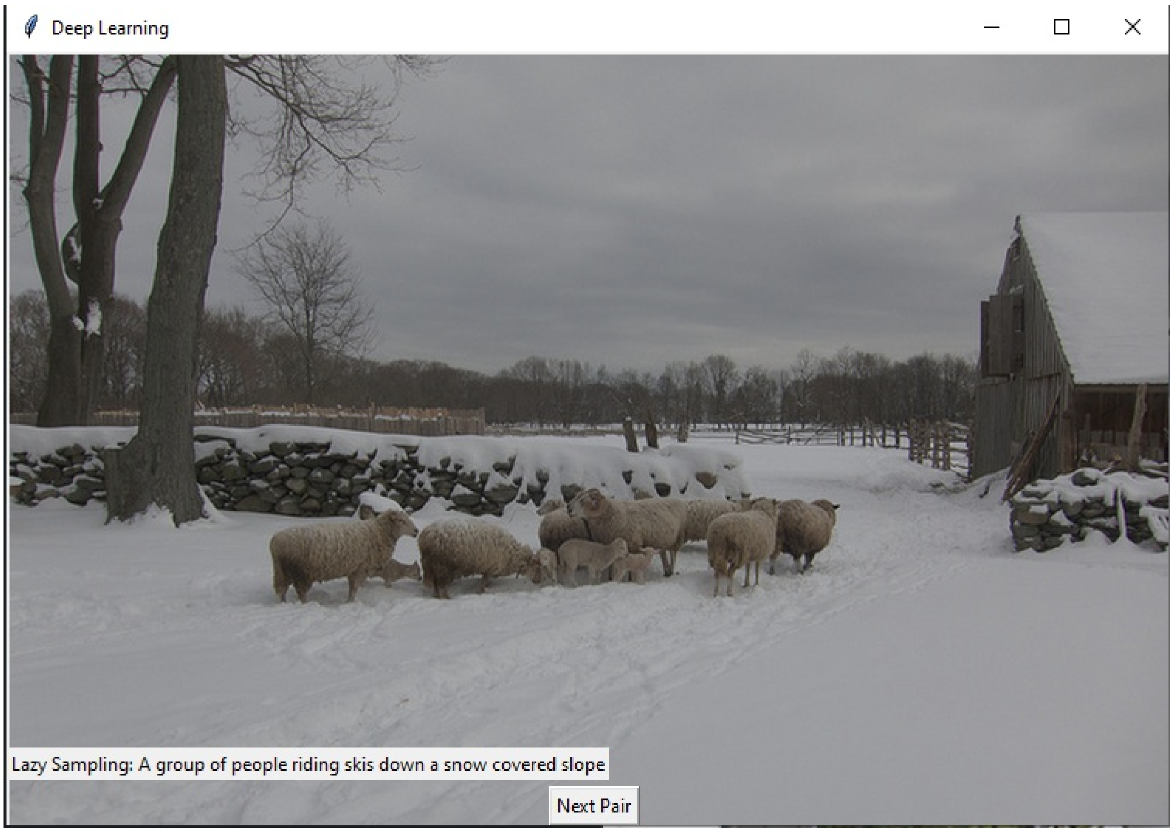
Some poor predictions:



Lazy Sampling: A zebra standing in a field with a large zebra .



Lazy Sampling: A stop sign that is on a street .



Lazy Sampling: A group of people riding skis down a snow covered slope

Group Members Contributions

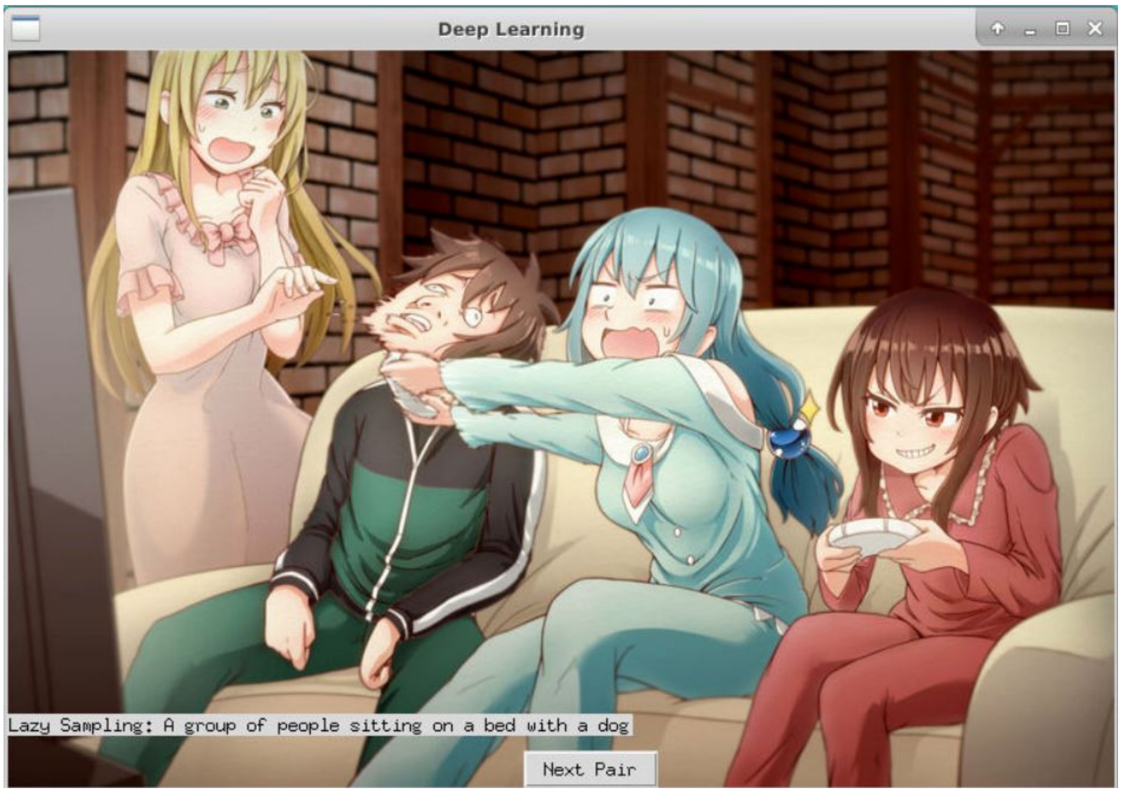
Gong Chen 1002870
Ivan Christian 1003056
Lim Theck Sean 1002777
Tang Mingzheng Paul 1002768

- Report : Everyone
- EncoderCNN : Ivan
- EncoderRNN : Gong Chen
- DecoderRNN : Ivan
- Generator : Paul
- Discriminator : Paul
- Train Function : Ivan
- Test Function : Gong Chen
- Custom Loss Function : Paul
- GUI : Sean

Weirdest/Funniest Captions

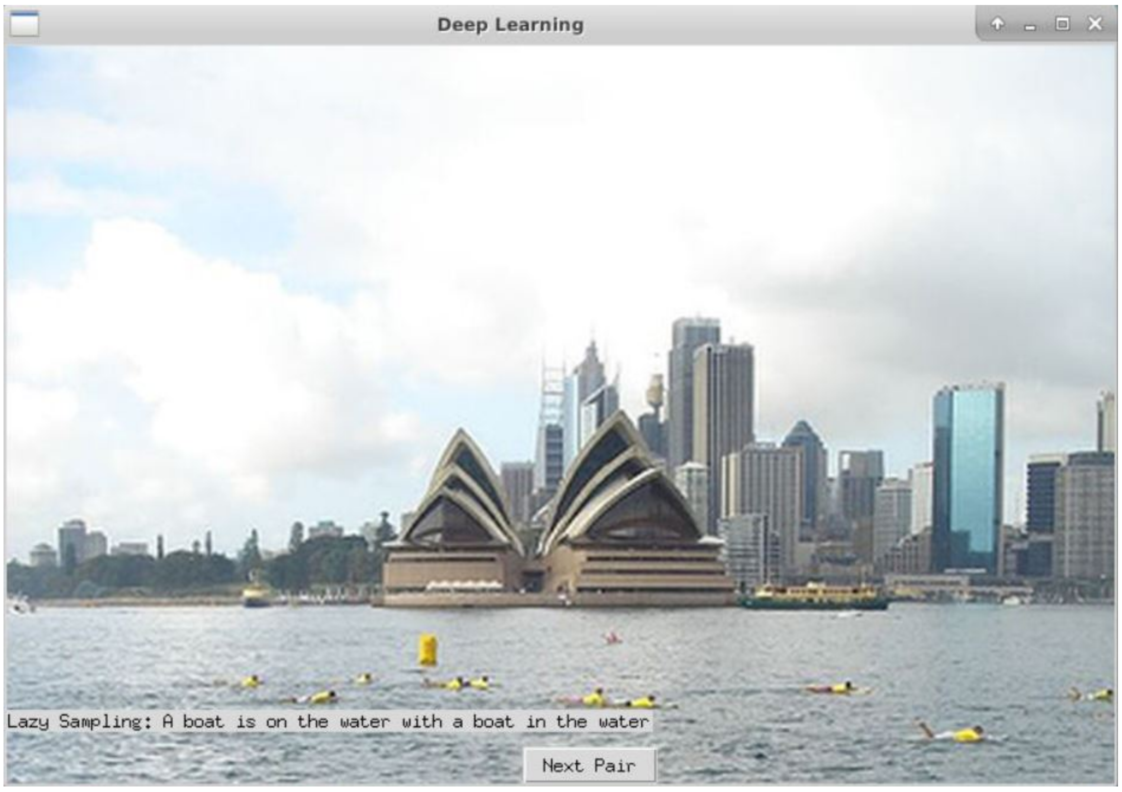
Our trained models can also be many different things!

1. An Otaku!



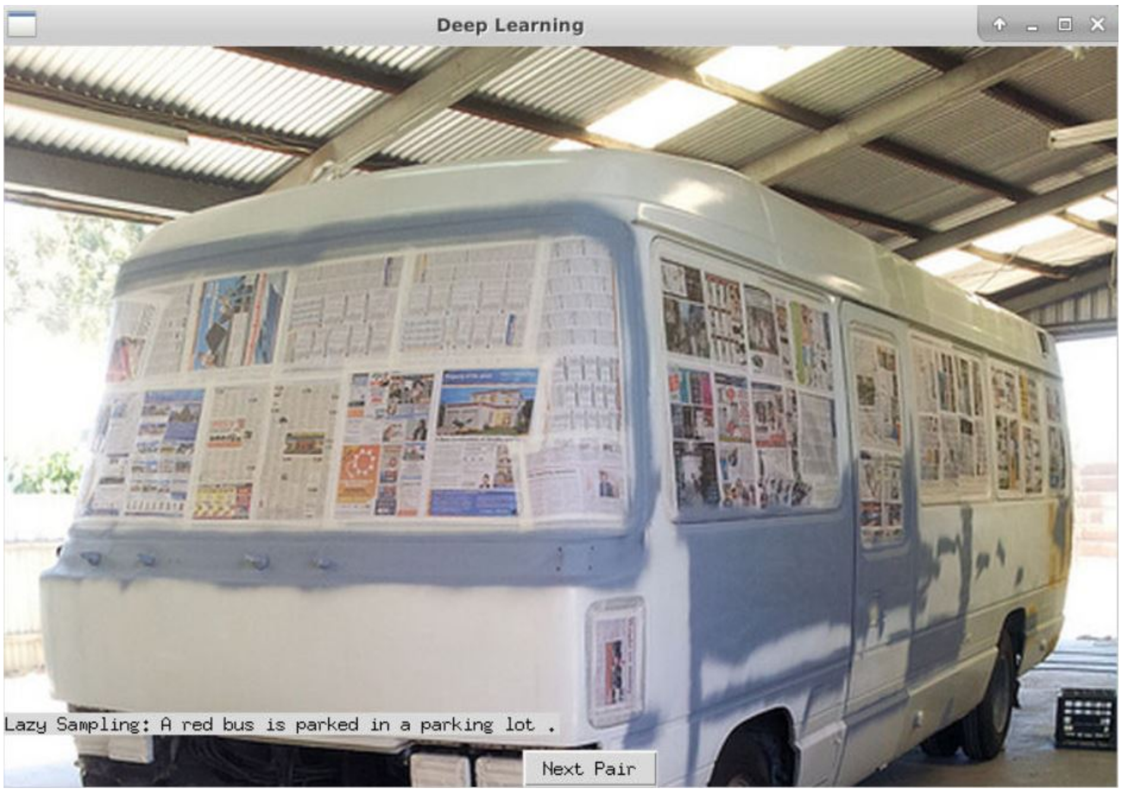
Lazy Sampling: A group of people sitting on a bed with a dog

2. An Architecture Connoisseur~



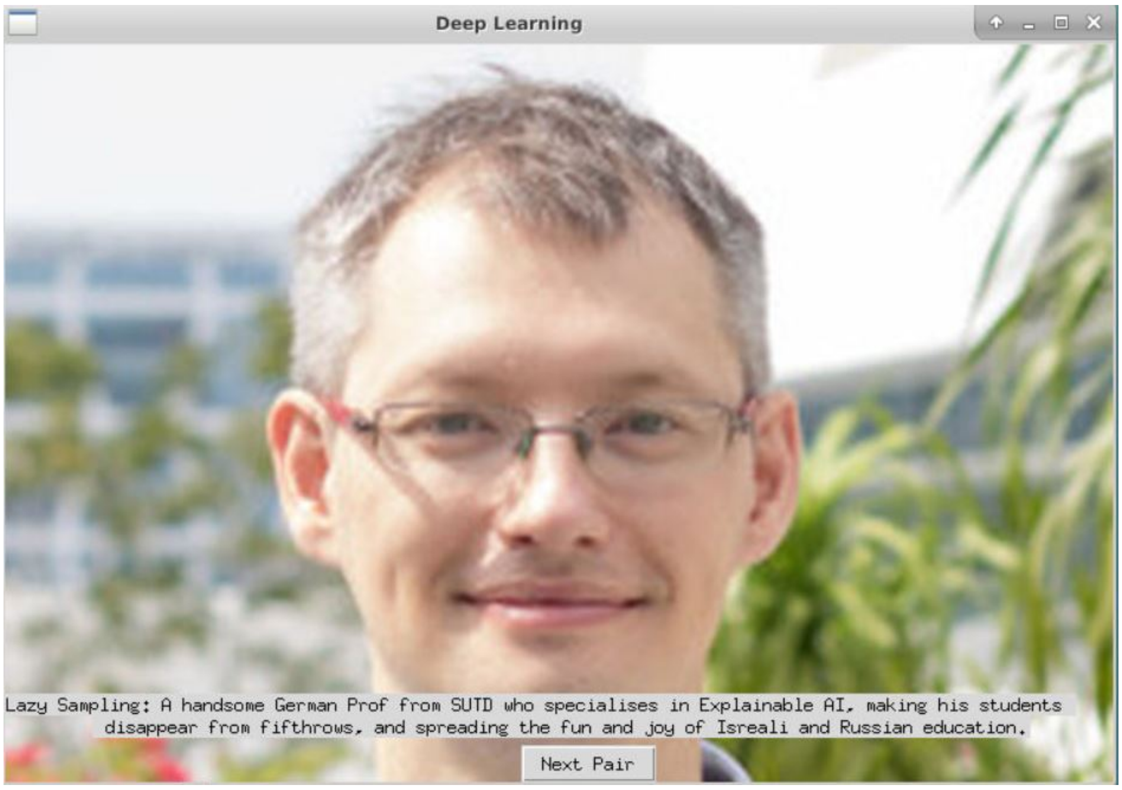
Lazy Sampling: A boat is on the water with a boat in the water

3. A Colour Blind >.<



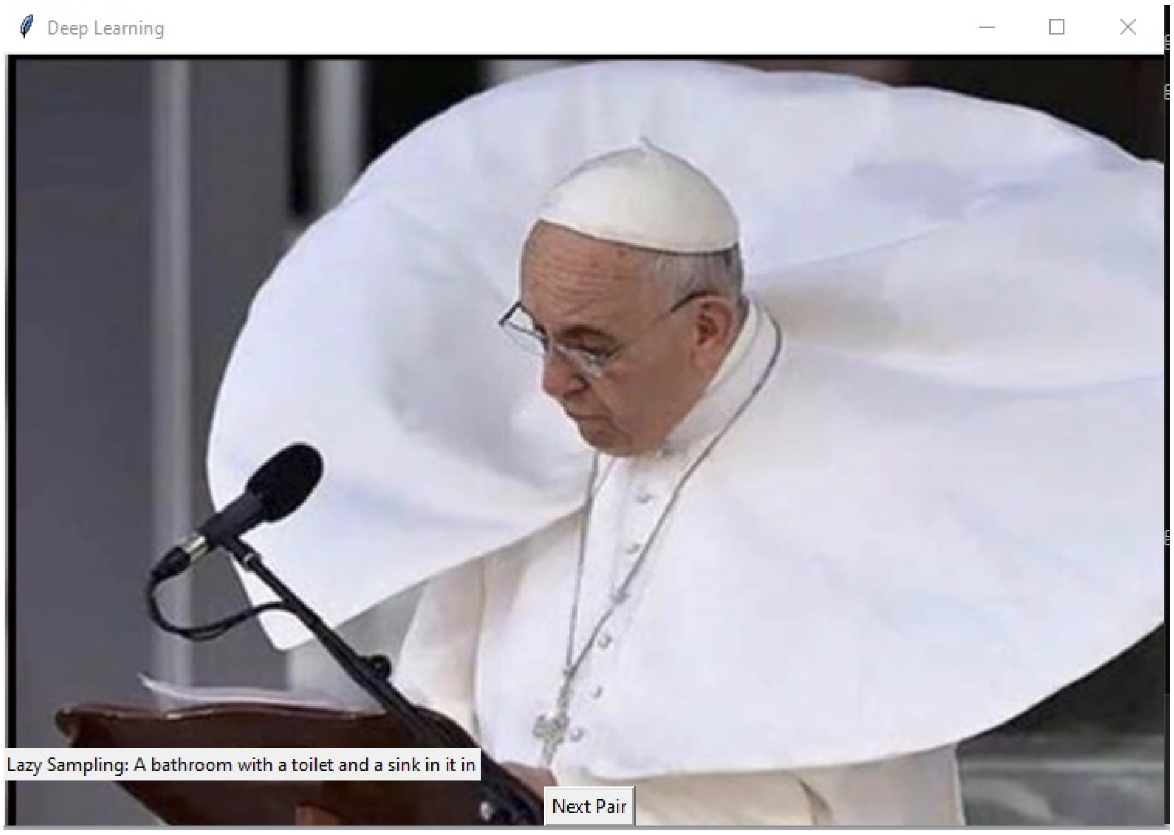
Lazy Sampling: A red bus is parked in a parking lot .

4. SUTD 50.039 Students…



Lazy Sampling: A handsome German Prof from SUTD who specialises in Explainable AI, making his students disappear from fifthrows, and spreading the fun and joy of Isreali and Russian education.

5. The Bathroom Pope



Lazy Sampling: A bathroom with a toilet and a sink in it in