Machine Learning(Deep Learning) Project Report

Project Title:

Hand-drawn Sketch recognition (Idea #3)

Problem Statement:

With the large number of images, we have nowadays, trying to find the exact one we want can be quite tricky, especially when we have something really specific in mind. In that case, using text description might be difficult. It can be extremely hard to express desired shape, pose, and viewpoint with natural language.

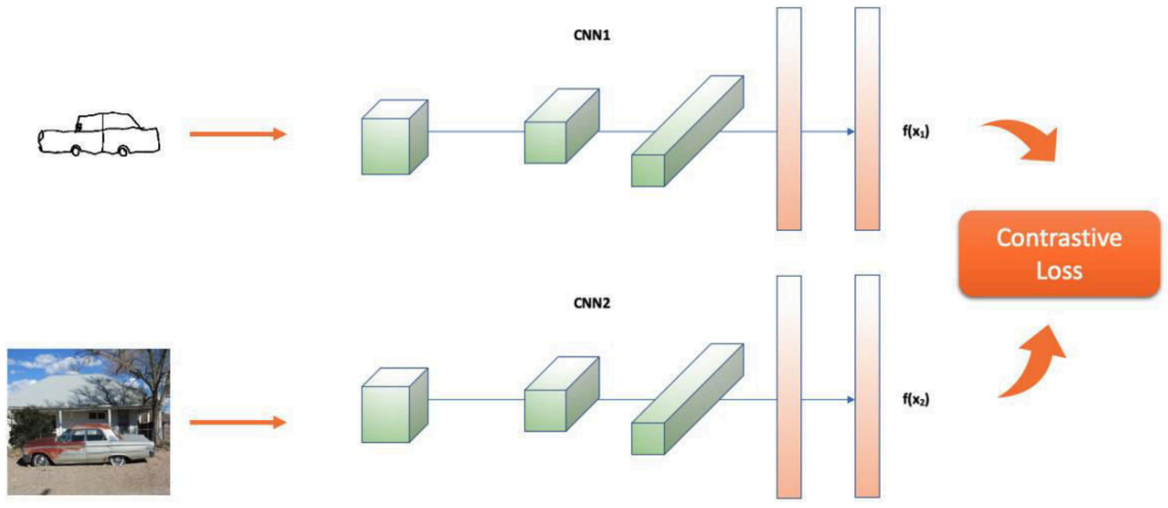
So instead of using text, a more intuitive approach is to use sketch to describe the image we're looking for. The goal here is to retrieve the most similar image, *not* to the sketch, *but* to some mental image that prompted the sketch.
In this project, the idea is to build a model which, given a sketch, can decipher what type of image that sketch might represent.

In this problem, we will be using Sketchy Database which contains a list of photos having 125 different categories, and sketches of the same 125 different categories

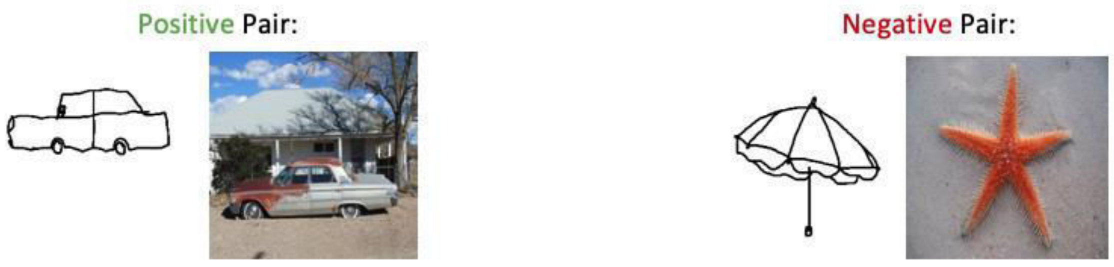
category	photo	sketch
airplane	100	709
alarm_clock	100	571
apple	100	561
.		.
.		.
wine_bottle	100	603
zebra	100	608
Total	12500	75481

Approach and Algorithm:

The architecture we have used for this problem is "Siamese Network". The idea is to have 2 CNN's(Convolution Neural Networks) with the exact same shared weights (and hence they are twins), and both of these CNN's are not classifiers, so they don't have any classification head, and hence they don't end in a SoftMax(which we generally use). Instead, the final layer is just a linear regression with no activation, which just outputs some feature vector.
So, if we input a set of 2 images (actual photo and a sketch of the same image), the output feature vectors would be very close to each other. On the other hand, if the 2 images are of different entities, the output feature vector would be very far from each other.



Each image in the image pair is fed to this network.



The network is optimised using a contrastive loss function (mentioned below).

Contrastive Loss:

Since we are using Siamese Architecture, we are not classifying the input images but we need a mechanism to differentiate between them. So, instead of using any classification loss function, this architecture is better suited to use a contrastive function. And in a nutshell, this function simply evaluates how well the network is able to distinguish between a given pair of images.

$$L_{contrast} = (1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\max(0, m - D_W)^2$$

where,

- Y ➔ If the inputs are from the same class, then the value of Y is 0 , otherwise Y is 1.
- D_w ➔ Euclidean distance between the outputs of the Siamese network
- m ➔ Margin value which is greater than 0. Having a margin indicates that dissimilar pairs that are beyond this margin will not contribute to the loss.

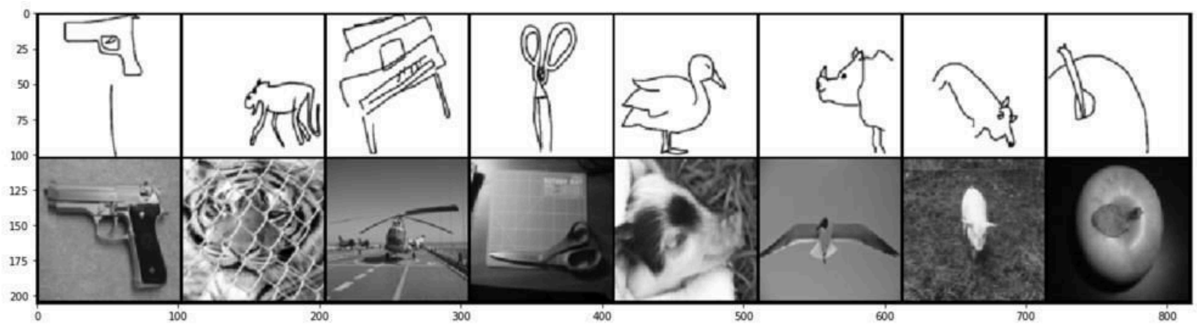
Data Preparation:

The idea we had is to make triplets of datasets, meaning, we will have a pair of images, like as follows:

- (Anchor, Positive, Match)
- (Anchor, Negative, Mismatch)

So, we created a custom dataset generator, where the function would generate pairs at random from the training dataset, and that pair could be a Positive pair or Negative pair. But to make sure there is some balance between positive and negative, we tried to generate around half of the images from the same class, while the other half is not.

Also, we reduced the dimension of the image to 100x100 and transformed it into grey scale.
We also split the whole dataset into train and test with 20% for testing and 80% for training.



```
[[0.],    [0.],     [1.],    [0.],    [1.],     [1.],     [0.],    [0.]]
```

Class_Labels = 0 (means matching) and 1 (means match is not found)

CNN Network:

Network used is a simple CNN, with:

Conv2d-> ReLU -> BatchNorm2d -> Conv2d -> ReLU -> BatchNorm2d -> Conv2d -> ReLU -> BatchNorm2d

Followed by a fully connected layer:

Linear -> ReLU -> Linear -> ReLU -> Linear

We used epochs of 200 with batch size of 64.

Training and Experiment:

We tried few of the experiments with different configurations by changing the no. of layers, epochs, batch size, different optimizers, etc. And finally, we used Adam optimizer, learning rate of 0.0005, our own custom contrastive loss function with a batch size of 64.

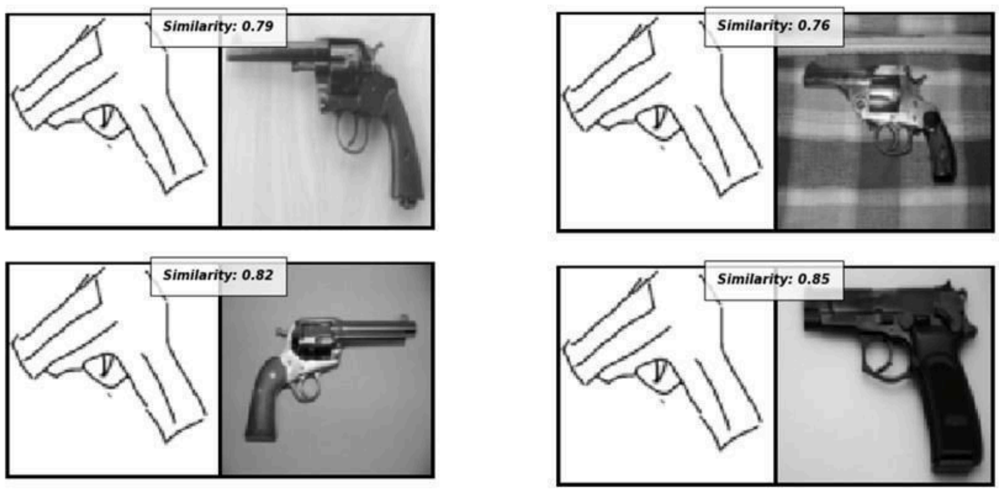
The code does the following for each pair of images, when it passes through the network:

1. The first image of a particular pair passes through the network.
2. The second image of that pair passes through the network
3. Compute the loss using the output from step#1 and step#2
4. Perform backprop and calculate the gradients.
5. Update the weights using an optimiser (Adam) and continue.

AWS Cloud platform was used to perform this training and test. The system used for the training was **ml.p3.16xlarge**, it comprised of **NVIDIA 8xV100 GPU and 64 CPUs**.

Testing and Inference:

Just like during training, for testing as well we carve out from the dataset, and we pick some random image of a sketch and try to find out the 10 (max) matches w.r.t the sketch. Given a sketch of a gun it shows what photo that sketch belongs to.



Saving the model and Inference:

Since the training took a lot of time, we thought to save the model weights, state dictionaries, etc. For this we used PyTorch's inbuilt feature of saving the model and loaded the model on a separate jupyter inference. So, in the inference jupyter-notebook, "Sketchy_Inference.ipynb" we loaded the model weights in a .pth file

Scope of Improvement and further exploration:

- Try with some standard network, like VGG, GoogleLeNet, etc.
- Experiment with few of their hyperparameters, dropout, etc.
- Come up with some other smart sampling technique for generating the triplet data instance.

Reference:

- Siamese Network – deeplearning.ai
- One Short Learning with Siamese Network
- The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies

Code:

[illegible]

Team Responsibility: