Machine Learning Engineer Nanodegree

Capstone Proposal

Mohammad Al-Fetyani

May 25, 2020

Abstract

This document is a project proposal to tackle the problem of image retrieval (CBIR) by presenting a content-based image retrieval approach. This approach utilizes the concept of Autoencoders to extract features from training images and compare them with a query image. Finally, it provides the top n similar images to the query image.

Domain Background

Image retrieval has always been an active area of research. In the past few years, the field of computer vision has grown exponentially due to the impressive progress in deep learning and the availability of an enormous amount of images with over a million images uploaded to the internet.

Image retrieval has caught the attention of big companies like Google and Amazon where you can search by providing an image instead of texts. There are various approaches to train deep learning models for this job. However, it can be a challenging task as some algorithms require a triplet to train on. This means that someone needs to manually label the enormous amount of images exist, which is difficult and definitely not recommended.

One of the works done on this subject is the use of deep ranking to retrieve similar images [3]. This approach is based upon the triplet loss where the images are manually organized into three groups. These groups are: given images, similar images to the given ones and any random images. The drawback of this approach is that it needs to manually label images which is not applicable in large datasets.

Problem Statement

The problem stems from the *AI Meets Beauty Challenge 2020* [1] where the problem is stated as follows:

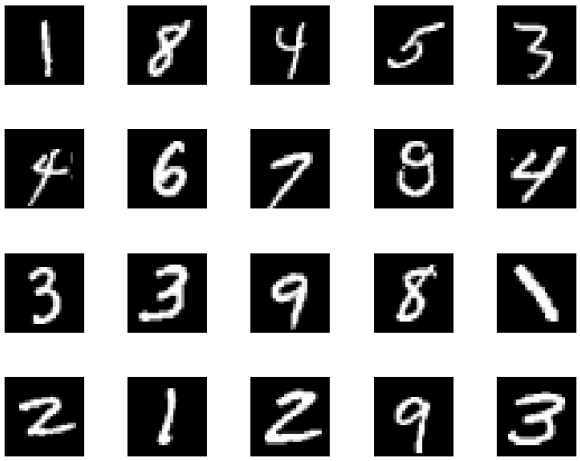
> *"Given a real-world image containing one beauty or personal care item, the task is to match the real-world example of this item to the same item in the Perfect-500K data set. This is a practical but extremely challenging task, given the limitation that only images from e-commerce sites are available in Perfect-500K and no real-world examples will be provided in advance."*

However in this project, we will handle more simpler problem since the dataset is huge and hard to manage. Given an image of a digit, return the closest images to it.

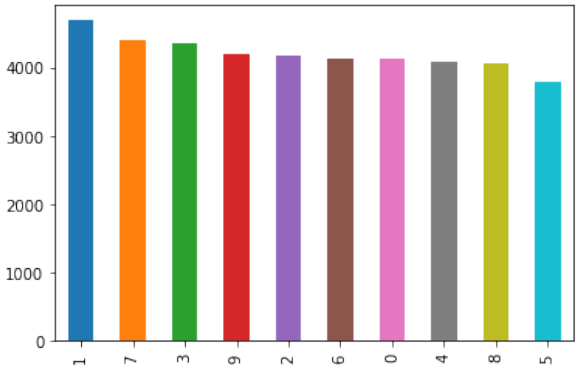
Datasets and Inputs

The MNIST dataset is appropriate for this problem because it can confirm the effectiveness of the proposed approach. The MNIST database consists of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

The images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. A sample of the dataset is provided in Figure 1a and the distributions of the digits in the dataset are provided in Figure 1b



(a) Sample of MNIST dataset.



(b) Distribution of the digits in the dataset.

Figure 1: Dataset exploration.

Benchmark Model

A benchmark model can be a simple linear regression model where we can compare the loss function of an Autoencoder to the loss function of the linear model. In addition, more advanced CNN models can be used with an Autoencoder to get better results.

Solution Statement

The approach proposed here trains an Autoencoder, which is considered an unsupervised model, to reconstruct input images. Once the Autoencoder is trained, we can extract the encoder of it and calculate the features of all images in the training set. Then, we compare the features of a given query image to the features of the training images and finally return the closest images.

Evaluation Metrics

This is an unsupervised problem where human evaluation is considered the best choice. However, since we have the labels of the images, we can use them to calculate the accuracy of a given model. That is,

for the top n images, calculate the number of images that are in the same class as follows:

$$\text{accuracy} = \frac{\sum_{\text{query images}} \text{number of correctly retrieved images}}{\sum_{\text{query images}} \text{number of all retrieved images}} \tag{1}$$

where we calculate the correctly retrieved images over all query images and then divide it by all retrieved images over all query images.

Project Design

Autoencoders work as follows:

1. Accept an input set of data (i.e., the input)
2. Internally compress the input data into a **latent-space representation** (i.e., a single vector that compresses and quantifies the input)
3. Reconstruct the input data from this latent representation (i.e., the output)

To build an image retrieval system with an autoencoder, what we really care about is that latent-space representation vector. Once an autoencoder has been trained to encode images, we can:

1. Use the encoder portion of the network to compute the latent-space representation of each image in our dataset — **this representation serves as our feature vector that quantifies the contents of an image**
2. Compare the feature vector from our query image to all feature vectors in our dataset (typically we would use either the Euclidean or cosine distance)

Feature vectors that have a smaller distance will be considered more similar, while images with a larger distance will be deemed less similar.

We can then sort our results based on the distance (from smallest to largest) and finally display the image retrieval results to the end user. This procedure is illustrated in Figure 2.

References

[1] AI Meets Beauty Challenge 2020,
`https://challenge2020.perfectcorp.com/`

[2] https://www.pyimagesearch.com/2020/03/30/autoencoders-for-content-based-image-retrieval-with-keras-and-tensorflow/

[3] Image Similarity using Deep Ranking,
`https://medium.com/@akarshzingade/image-similarity-using-deep-ranking-c1bd83855978`

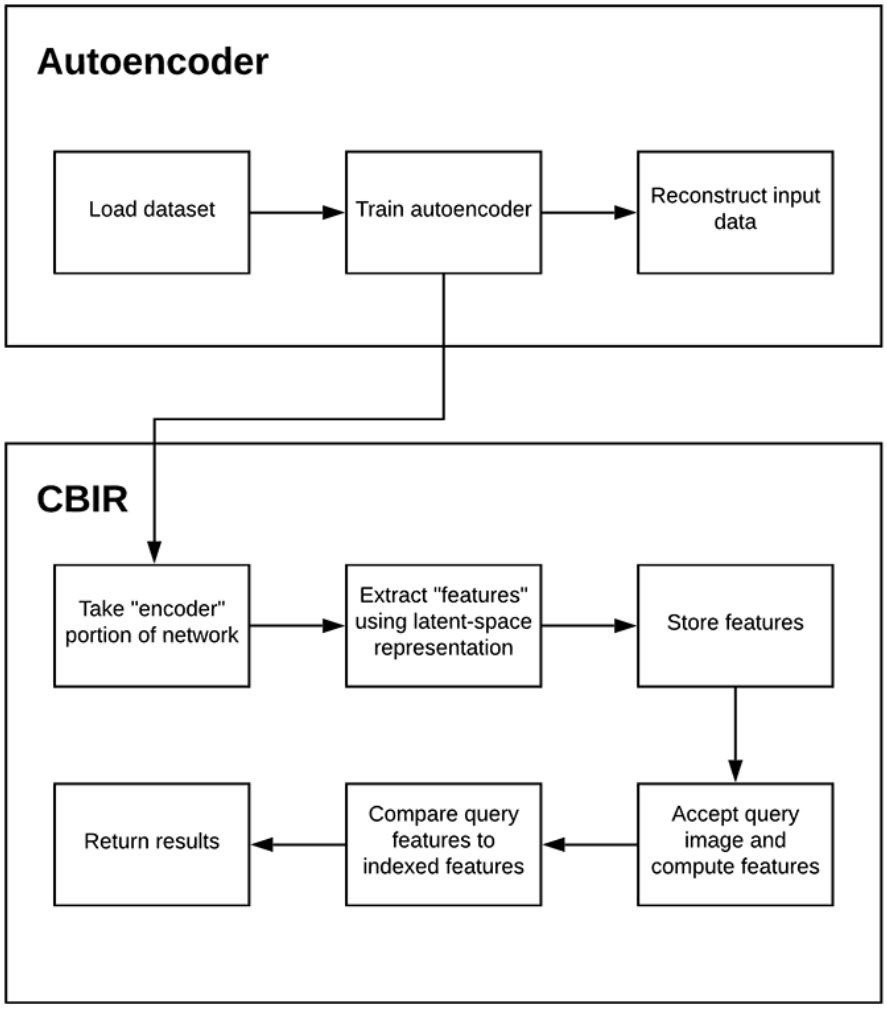


Figure 2: Procedures of image retrieval using autoencoders. [2]