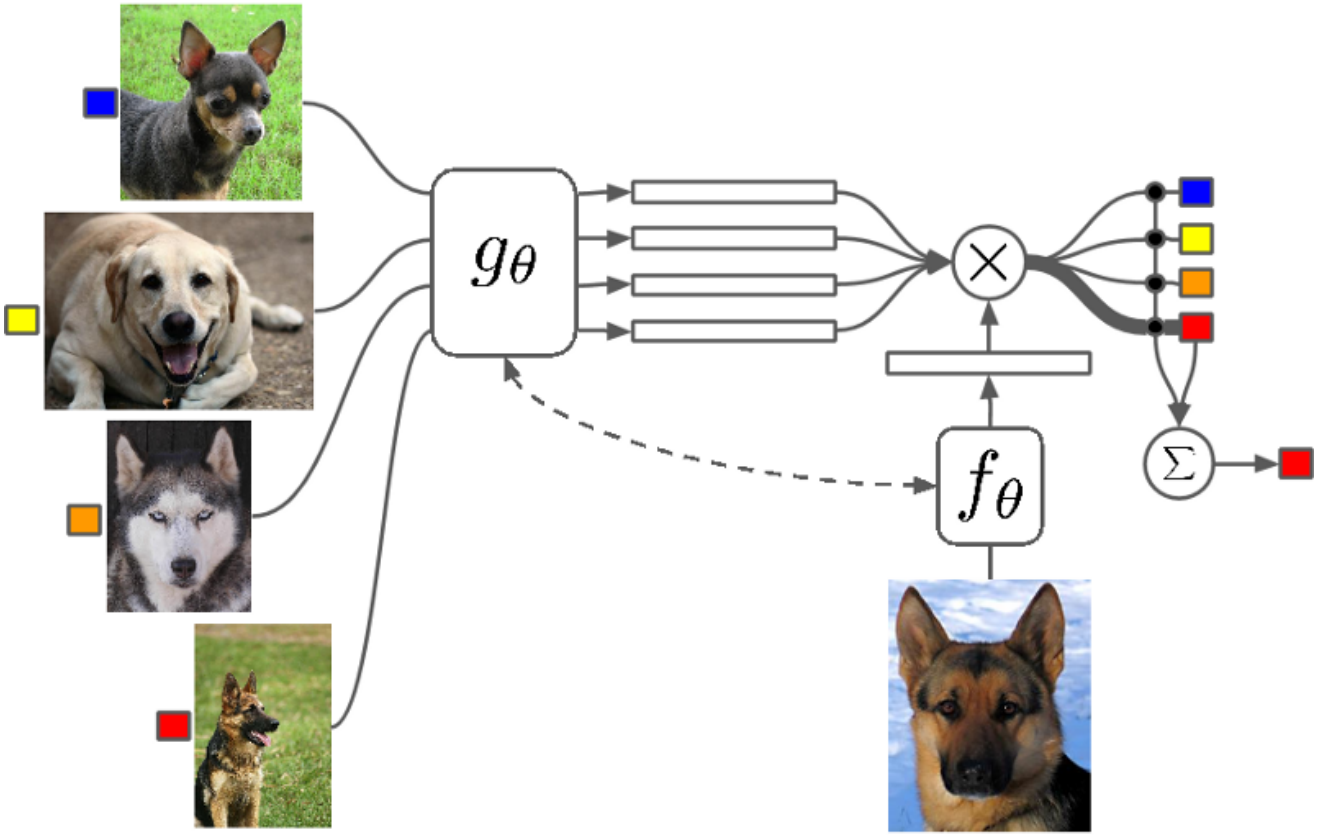
One Shot Learning with Siamese Networks in PyTorch



Source

> This is Part 1 of a two part article. You can read part 2 here

Deep neural networks are the go to algorithm when it comes to image classification. This is partly because they can have arbitrarily large number of trainable parameters. However, this comes at a cost of requiring a large amount of data, which is sometimes not available. I will discuss One Shot Learning, which aims to mitigate such an issue, and how to implement a Neural Net capable of using it ,in PyTorch.

*This article assumes some familiarity with **neural networks**.*

How this article is Structured

This is a two part article. I will go through the **theory** in **Part 1** , and the PyTorch **implementation** of the theory in **Part 2**.

This article takes cues from this paper.

Standard Classification vs. One Shot Classification

Standard classification is what nearly all classification models use. The input is fed into a series of layers, and in the end , the class probabilities are output. If you want to predict dogs from cats, you train the model on similar(but not same) dogs/cats pictures that you would expect during prediction

time. Naturally, this requires that you have a dataset that is similar to what you would expect once you use the model for prediction.

One Shot Classification models, on the other hand, requires that you have just **one training example** of each class you want to predict on. The model is still trained on several instances, but they only have to be in the similar domain as your training example.

A nice example would be facial recognition. You would train a One Shot classification model on a dataset that contains various angles , lighting , etc. of a few people. Then if you want to recognise if a person X is in an image, you take one single photo of that person, and then ask the model if that person is in the that image(*note, the model was not trained using any pictures of person X*).

As humans, we can recognize a person by his/her face by just meeting them once, and it is desirable by computers because many times data is at a minimum.

Enter Siamese Networks

Siamese networks are a special type of neural network architecture. Instead of a model learning to *classify* its inputs, the neural networks learns to *differentiate* between two inputs. It learns the similarity between them.

The architecture

A Siamese networks consists of two identical neural networks, each taking one of the two input images. The last layers of the two networks are then fed to a contrastive loss function , which calculates the similarity between the two images. I have made an illustration to help explain this architecture.

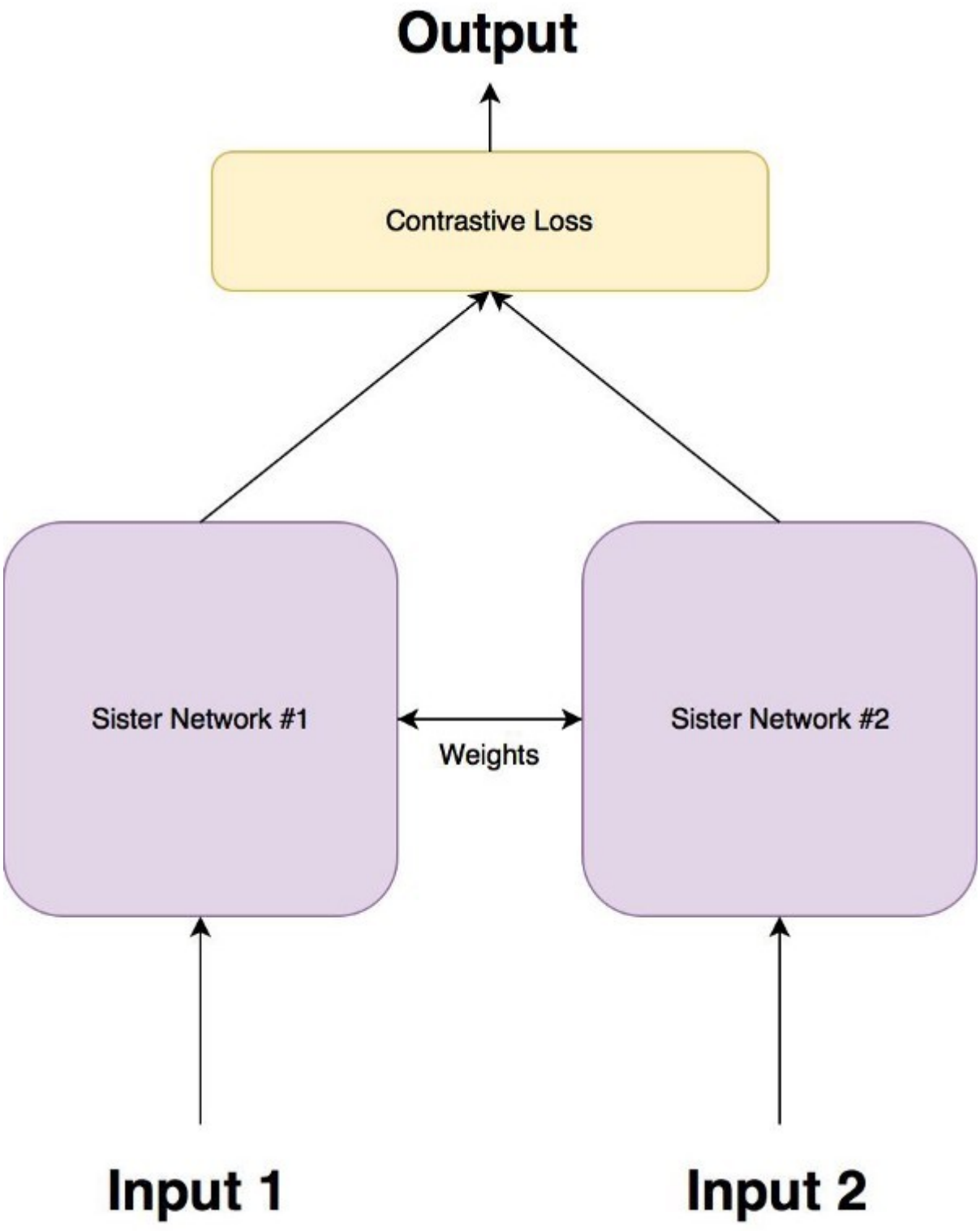


Figure 1.0

There are two sister networks, which are identical neural networks, with the exact same weights.

Each image in the image pair is fed to one of these networks.

The networks are optimised using a contrastive loss function(we will get to the exact function).

Contrastive Loss function

The objective of the siamese architecture is not to classify input images, but to *differentiate* between them. So, a classification loss function (such as cross entropy) would not be the best fit. Instead, this architecture is better suited to use a contrastive function. Intuitively, this function just evaluates how well the network is distinguishing a given pair of images.

You can read more details here

The contrastive loss function is given as follows:

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

where ***Dw*** is defined as the euclidean distance between the outputs of the sister siamese networks. Mathematically the euclidean distance is :

$$\sqrt{\{G_W(X_1) - G_W(X_2)\}^2}$$

where ***Gw*** is the output of one of the sister networks. **X1** and **X2** is the input data pair.

Equation 1.0 Explanation

Y is either 1 or 0. If the inputs are from the same class , then the value of Y is 0 , otherwise Y is 1

max() is a function denoting the bigger value between **0** and **m-Dw**.

m is a **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. This makes sense, because you would only want to optimise the network based on pairs that are actually dissimilar , but the network thinks are fairly similar.

The Datasets

We will use two datasets , the classic MNIST , and OmniGlot. MNIST will be used to train the model to understand how to differentiate characters, and then we will test the model on OmniGlot.

OmniGlot Dataset

The OmniGlot Dataset consists of examples from 50 international languages. Each alphabet in each language has 20 examples only. This is considered a ‘transpose’ of MNIST, where the number of classes are less (10), and the training examples are numerous. In OmniGlot, there are a very large number of classes, with few examples of each class.

OmniGlot will be used as our on shot classification dataset, to be able to recognise many different classes from a handful of examples only.

Conclusion

We glossed over the general premise of one shot learning, and trying to solve it using a neural network architecture called Siamese Network. We talked about the loss function that distinguishes between an input pair.

In part 2 of this article , we will go through implementing such an architecture, and training it on MNIST, then predicting it on OmniGlot.

P.S.

If you liked this article , be sure to leave a ❤. It lets me know I am helping. Continue on to part 2, which covers the implementation of discussed ideas.

Continue on to part two :

Facial Similarity with Siamese Networks in PyTorch