Image classification with self-supervised learning

1. Introduction

In cases when there is little data available the process of learning good features for machine learning applications may become expensive in terms of computation and hard to prove. Historically, deep learning algorithms don't work well if you have only a few training examples. The latter is the interesting and fun part for our project, although we will work on a dataset which is well-known and a lot of implemented jobs already exist on it, we are going to work on a small number of training examples. Few-Shot learning can be used to improve many existing image recognition algorithms. Also Few-Shot learning can be used for most face recognition systems because you might have only one picture of each person in a database. And on each update of a new person retraining our known convolutional networks does not seem a good approach.
In this project we will do image classification given only a few examples of each novel class we must correctly make predictions. The goal is to attain reasonable accuracy with as few training examples as 4 per class.

2. Dataset

We will be using MNIST(Modified National Institute of Standards and Technology database) dataset. It is a large dataset commonly used in fields of deep learning and machine learning to train different image processing systems. It contains 60,000 images for training and 10,000 images for testing. However, we are not going to work with this particular dataset; in our customized version we will be having only 4 training images per class.

3. Algorithm step by step

- Getting the splitted data.
- Building a network,with two convolutional layers, max pooling layer, dropout regularizations and for a given image returning a vector.
- Building a model for training which computes the loss layer for given triplets .
- Compiling the model with adam optimizer and `0.00007`.
- Getting a random batch of triplets from train data.
- Getting a hard batch of triplets from the random batch.

- Training the network with different hard batches.
- Getting the vector for given images from the test set, from each class of train images randomly getting a representative images and the vectors of them, calculating distances between vectors of input images and representative images. Returning list of indexes; each index shows minimal distance between each input image vector and representative vectors.

Let's discuss these steps:

Siamese neural networks and One-Shot Categorization

A siamince neural network consists of twin networks, which accept distinct inputs, but are joined by a distance function. We learn image representation with siamese neural networks, then reuse that networks features for one-shot learning. In case of one-shot learning we may only observe a single example of each possible class before making a prediction about the test instance. In particular, it learns a function which inputs two images and outputs the degree of difference between the two images. So if the two images are of the same class, we want this to output a small number. And if the degree of difference between them is less than some threshold, then we would predict that these two pictures are of the same class and if it is greater than threshold they belong to different classes. Passing two input images to a sequence of convolutional and pulling and fully connected layers, ending up with a feature vector, the difference function will be calculated as the euclidean distance of those two vectors.
So our strategy is to pairise each novel class representative with a test sample. The highest probability for the one-shot task (the lowest distance function) would decide to which class the picture belongs.

So the parameters of our network define an encoding as a vector of length 10.

Triplet loss function

To apply the triplet loss function we need to compare pairs of images. Lets define a triplet. It is a triplet of images Anchor, Positive, Negative, where Anchor and Positive are pictures of the same class and Negative is a representative of another class. To learn the parameters of the network for a given Anchor and Positive we need their encodings to be similar, because they are of the same class, for Anchor and Negative to be different. So we want $(dist\,(A,P) - dist\,(A,N)) \leq 0$.
But in this case the network can learn to output 0-s as $0 - 0 \leq 0$. To prevent this we say $dist(A,P) - dist(A,N) + margin \leq 0$. In our model margin is 0.2.

Triplet loss function is defined on triplets of images. For given triplet $A,\ P,\ N$ the loss is :
$L(A,P,N)\ =\ max(dist(A,P)\ -\ dist(A,N)\ +\ margin\ ,\ \ 0)$.
We take max, because as long as $dist(A,P)\ -\ dist(A,N)\ +\ margin\ \leq\ 0$ our network doesn't care how far the triplets encodings are from each other.
Overall cost function $J = \sum_{i=1}^{m} (A^{(i)}, P,^{(i)} N^{(i)})$.

Let's discuss how we choose these triplets for our network:

Our train data consists of 10 classes, and we have 4 images, labels (chosen randomly from real dataset) per each class. First we choose random triplets to study: We choose a random class, from that random class we choose 2 random pictures: One as an Anchor, the other as Positive, then we choose another random class and a random picture from that class as Negative. This means that our data have $10 * C_4^2\ *\ 9 * \ 4\ \ =\ 2160$ possible triplets.
When we choose random triplets, the condition for distances is easy to satisfy. So our network will not learn much from them. Thus, for our batch of random triplets we choose hard triplets (these are the triplets with $d\,(A,N)\ <\ d\,(A,P)$ and with the same amount of random triplets from the remaining ones of the batch. From large sizes of random batches we chose mach smaller sizes of hard batches. In this case the learning algorithm will try hard.
After applying triplett loss technique we are getting two times higher accuracy: from 0.34 to 0.68 on test data.

Conclusion

So to summarize, our algorithm builds a network, which learns on hard triplets and computes optimal encoding for each image, then for a given test image its encoding is predicted. We predict encodings of our training set samples. After we compute distances between test image encoding and encodings of training set images. The index of minimal distance class will be the index of test image class.

Resources

https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d

https://medium.com/@crimy/one-shot-learning-siamese-networks-and-triplet-loss-with-keras-2885ed022352

https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

https://arxiv.org/abs/2001.07685