Siamese Neural Networks for One-shot Image Recognition

Harsh Sulakhe: 2018A7PS0186P
Shreyans Jain: 2018A7PS0253P
Pranav Gupta: 2018A7PS0190P

Title and Affiliation of Authors

Siamese Neural Networks for One-shot Image Recognition

Authors:

- Gregory Koch (*Department of Computer Science, University of Toronto*)
- Richard Zemel (*Department of Computer Science, University of Toronto*)
- Ruslan Salakhutdinov (*Department of Computer Science, University of Toronto*)

Link to the original paper

Aim

The following were the broad aims of the paper:

- Develop a model that can classify/identify images belonging to a new class based on just a single example. This technique is referred to as *one-shot classification*.
- Utilise a deep siamese convolutional neural network for the task.
- Train the model so that it can rank the similarity of image-pairs, hence allowing us to classify every new image into its correct class.
- Develop a system that can beat existing architectures and maintain acceptable performance so that it can be replicated at an industrial scale

Methodology

The authors utilise a class of feed-forward neural networks called Siamese neural networks for this task. These networks convert 2 input images into their feature vectors using convolutional layers and calculate a similarity metric on the feature vectors. In an actual test scenario, one of the 2 input images is the image we want to classify, and the other is one from a set of images that the network has been trained on. This way, the network can generalise easily and even classify images that it has never seen before with just one training example.

The paper trains a Convolutional Siamese Net on the Omniglot dataset which contains handwritten characters from the alphabets of various less-spoken languages. Further, the model is tested against the MNIST dataset to gauge how well it can handle one-shot trials with completely different images.

Outcomes

As reported in Table 1, the proposed Siamese Conv. Net achieves around 92% test accuracy on the Omniglot dataset. This significantly beats all other deep learning based approaches. However, the model still remains behind HBPL (95% test accuracy). Further, the model generalises well on the MNIST dataset with an accuracy of about 70%. Overall, the model proves to be robust enough to be implemented on an industrial scale with larger datasets.

Method	**Accuracy (%)**
30k examples	91.90
90k examples	93.15
150k examples	93.42

Table-1 Accuracies with different number of training examples

Method	**Accuracy (%)**
Human	95.5
HBPL	95.2
Siamese Conv. Net	92.0
Hierarchical Deep	65.2

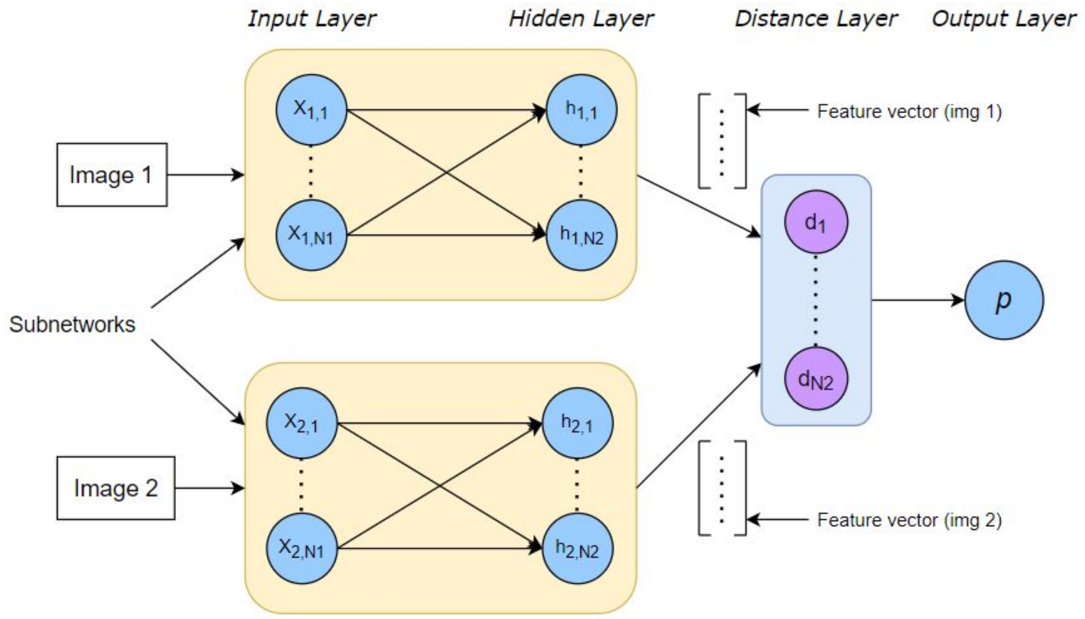
Table-2 Accuracies of various models on the same task

Background

- Siamese Neural Networks
- Use of Convolutional Layers
- About HBPL

Siamese Neural Networks

- A siamese neural network is a combination of 2 *twin sub-networks* (both accepting different inputs) that cascade into a single, fully-connected layer at the top, giving the network's output.
- The twin sub-networks are said to be *tied*, meaning they share the same weight matrices and parameters. Because the parameters are shared, each sub-network performs the same computation on both inputs.
- Siamese nets were designed with the purpose of ranking the similarity between the 2 inputs (primarily images). This is done by encoding the images into their feature vectors and comparing these vectors using a distance function (eg. L1/L2 distance)
- Siamese networks accurately predict how similar its inputs are, allowing us to classify new classes of data without having to re-train the network (which is usually the case when adding/removing new classes)



Siamese NN Architecture

The network has L layers with N_L units each. $h_{1,l}$ represents hidden layer unit for layer l in the first twin and $h_{2,l}$ represents the same for the second twin. The network applies ReLU activation in the first L-2 layers and sigmoid activation in the remaining. *The encoding of images is done using convolutional layers (next slide).*

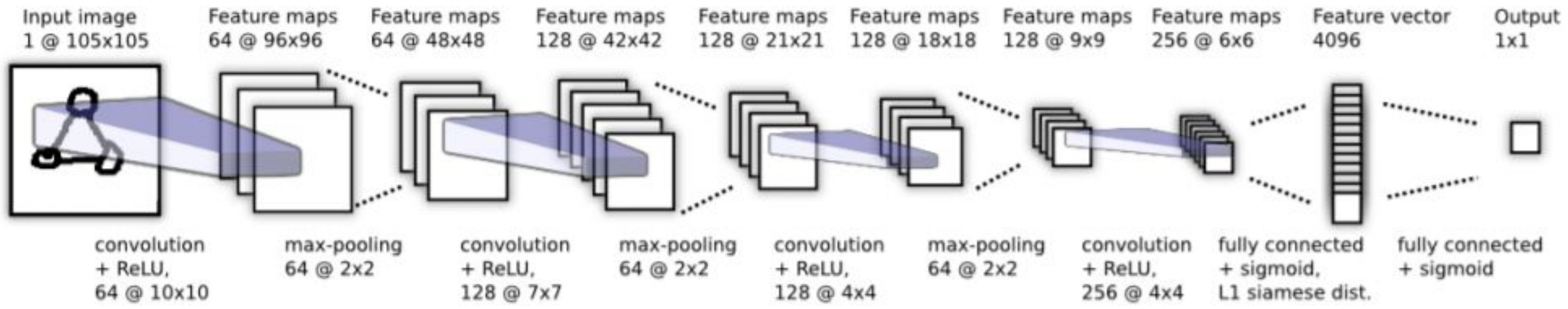
Each sub-network outputs a single dimensional feature vector that is fed to the distance layer. The distance between the 2 twins' vectors is calculated and fed into a sigmoidal activation function. The prediction vector, **p**, is given by:

$$\mathbf{p} = \sigma\left(\sum_j \alpha_j \,|\, \mathbf{h}^{(j)}_{1,L-1} - \mathbf{h}^{(j)}_{2,L-1} \,|\right)$$

Use of convolutional layers

Inside each sub-network is a standard CNN architecture that converts an RGB image into a one-dimensional feature vector. This model uses the traditional Convolution-Max Pooling-Fully Connected layer stack. The feature vector is then fed to the L1-distance layer (common among the 2 sub-networks) for calculation of the similarity metric.

The architecture of the convolutional layers is as shown below:



Use of convolutional layers

Each convolutional layer uses a single channel with filters of varying sizes and fixed stride of 1. The number of convolutional filters is taken as a multiple of 16, for optimization purposes. ReLU activation is applied on the output of every convolutional layer, optionally followed by a max-pooling layer with filter size and stride of 2. The governing equation for each filter is :

$$a_{1,m}^{(k)} = \text{max-pool}(\max(0, \mathbf{W}_{l-1,l}^{(k)} \star \mathbf{h}_{1,(l-1)} + \mathbf{b}_l), 2)$$
$$a_{2,m}^{(k)} = \text{max-pool}(\max(0, \mathbf{W}_{l-1,l}^{(k)} \star \mathbf{h}_{2,(l-1)} + \mathbf{b}_l), 2)$$

The weights of the $W_{l-1,l}$ tensor are learnt using a standard cross-entropy loss function, where the output vector y is 1 when inputs images are similar and 0 when not. The loss function is given by:

$$\boldsymbol{L}(x_1^{(i)}, x_2^{(i)}) = y(x_1^{(i)}, x_2^{(i)}) \log \mathbf{p}(x_1^{(i)}, x_2^{(i)}) + (1 - y(x_1^{(i)}, x_2^{(i)})) \log (1 - \mathbf{p}(x_1^{(i)}, x_2^{(i)}))$$

Hierarchical Bayesian Program Learning

HBPL (proposed by Lake et al) is an approach to one-shot learning that achieves near human-level accuracy on the task of one-shot classification.

The original paper presents a generative model that can build whole objects from individual parts. It outperforms most deep-learning based models by imposing a strong inductive bias on the classifier, meaning even a single favorable example can cause the algorithm to steer towards the solution. *One of the aims of the SNN paper was to beat HBPL at this task.*

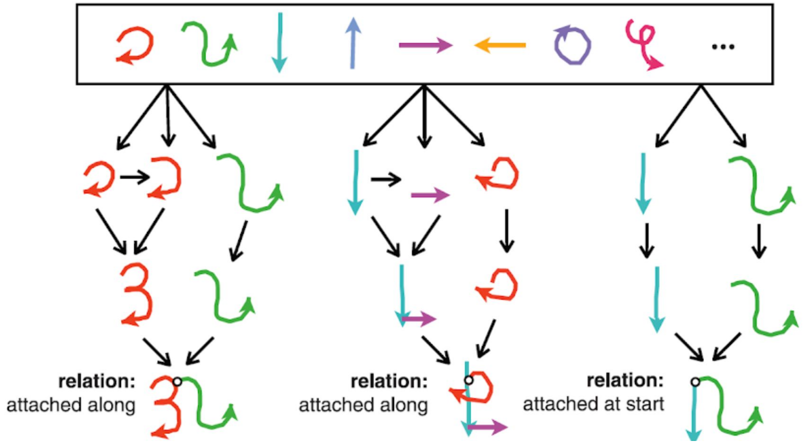


Fig. Images are broken down into individual objects that are then re-assembled to identify a particular character

Dataset

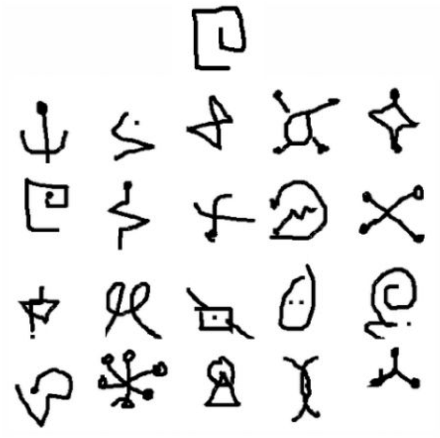
- The authors train their model using the Omniglot dataset.
- The Omniglot data set is designed for developing more human-like learning algorithms. It contains 1623 different handwritten characters from 50 different alphabets. Each of the 1623 characters was drawn online via Amazon's Mechanical Turk by 20 different people.
- It was collected by Brenden Lake and his collaborators at MIT.
- Each character in Omniglot is a 105x105 binary-valued image

Aurek-Besh
Futurama
Greek
Hebrew
Korean
Latin
Malay
Sanskrit

The Omniglot dataset contains a variety of different images from alphabets across the world.

Dataset: Omniglot

- The number of letters in each alphabet varies considerably from about 15 to upwards of 40 characters. All characters across these alphabets are produced a single time by each of 20 drawers.
- The data is split into a 40 alphabet background set and a 10 alphabet evaluation set.
- The background is used to learn general knowledge about characters (e.g., feature learning, meta-learning, or hyperparameter inference).
- One-shot learning results are reported using alphabets from the evaluation set.


Example of a 20-way one-shot classification task using the Omniglot dataset. The lone test image is shown above the grid of 20 images representing the possible unseen classes that we can choose for the test image. These 20 images are our only known examples of each of those classes.

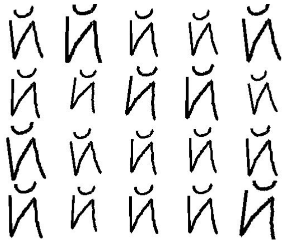
Dataset: Augmentation

As described in the paper, we augmented the training set with small affine distortions. For each image pair $\mathbf{x_1}, \mathbf{x_2}$, we generated a pair of affine transformations $\mathbf{T_1}, \mathbf{T_2}$ to yield $\mathbf{x'_1} = \mathbf{T_1(x_1)}, \mathbf{x'_2} = \mathbf{T_2(x_2)}$ where $\mathbf{T_1}, \mathbf{T_2}$ are determined stochastically by a multidimensional uniform distribution.
So for an arbitrary transform **T**, we have,

$\mathbf{T} = (\boldsymbol{\theta}, \boldsymbol{\rho x}, \boldsymbol{\rho y}, \mathbf{sx}, \mathbf{sy}, \mathbf{tx}, \mathbf{tx})$, with $\boldsymbol{\theta} \in [-10.0, 10.0]$, $\boldsymbol{\rho x}, \boldsymbol{\rho y} \in [-0.3, 0.3]$, $\mathbf{sx}, \mathbf{sy} \in [0.8, 1.2]$, and $\mathbf{tx}, \mathbf{ty} \in [-2, 2]$

Each of these components of the transformation is included with probability 0.5.

A sample of random affine distortions generated for a single character in the Omniglot data set.
(*as shown in the paper*)



An example of random affine distortions generated by our own implementation of the paper.
(*not present in the paper*)

Implementation

Framework : PyTorch 1.4.0
CUDA version : 10.1
GPU : NVIDIA GeForce GTX 1660Ti

File Descriptions:

1. **train.py** - Consists of the main training code
2. **mydataset.py** - Consists of custom datasets used for training and testing
3. **plot_loss.py** - Code to plot and save the graph for training loss v/s iterations
4. **affine.py** - Code to create and save affine transformations of custom images, with and without random sampling.
5. **model.py** - Consists of the main model architecture of the convolutional siamese twins.
6. **evaluate.py** - Code to evaluate the pre-trained model on 20-way one shot trials.

Implementation: Pseudocode

The main training loop works as follows:

For every iteration:

1. Sample batch_size number of pairs from the training set with their corresponding label.
2. Run both of the images through their respective twin within the model.
3. Calculate loss (binary cross entropy) based on output score of Siamese network.
4. Update network weights using gradient of this loss.

After a fixed number of iterations (we used 200), test the model to see if it’s learning, on the evaluation set.

Implementation: Pseudocode

To evaluate the model we did the following:

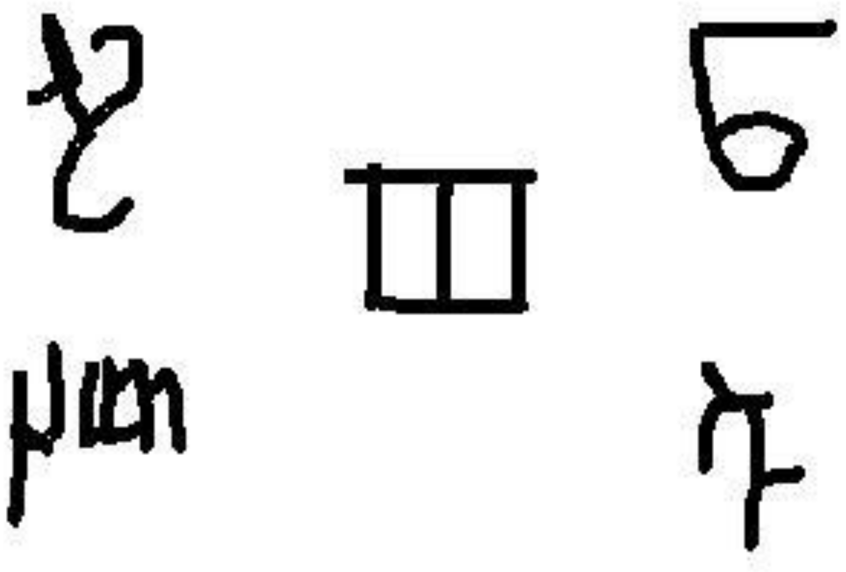
1. Sample 20 pairs (for 20-way one shot learning) from the evaluation set, 400 times.
2. For each time, check if the model produces the closest similarity score for the pairs in question.
3. If the prediction matches the true pair, increase number of correct predictions by 1, else increase number of incorrect predictions by 1.

Results and Discussion

The final accuracy on the evaluation set, consisting of 20 alphabets was calculated using 400 random 20-way one shot trials as described in the paper.

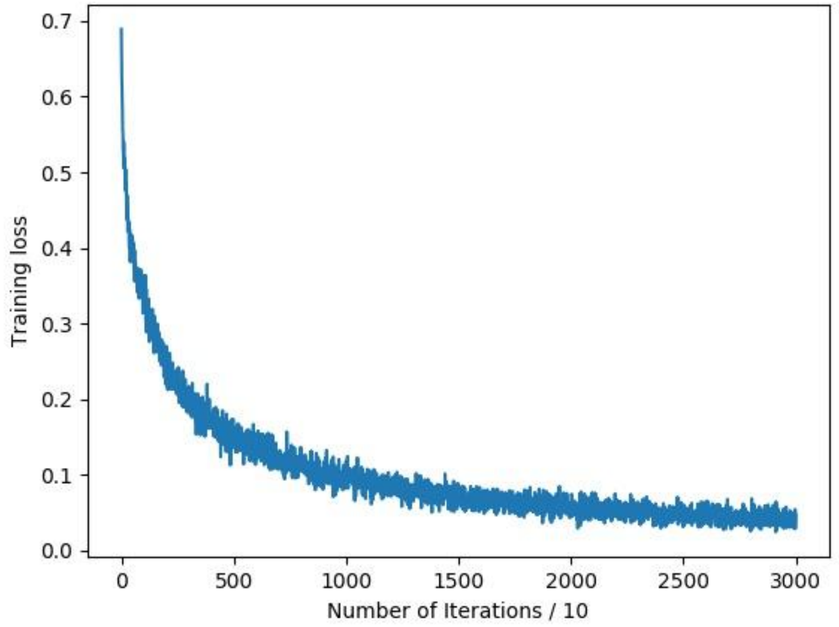
To evaluate our model, we ran multiple such tests. Compared to the authors’ best accuracy of 92%, we obtained a **minimum accuracy of 87.25%** and a **maximum accuracy of 91.75%** over the multiple tests we ran.

Results and Discussion



Few images that the model correctly classifies one-shot

Results and Discussion



Training loss *v/s* iterations
(*In our implementation*)

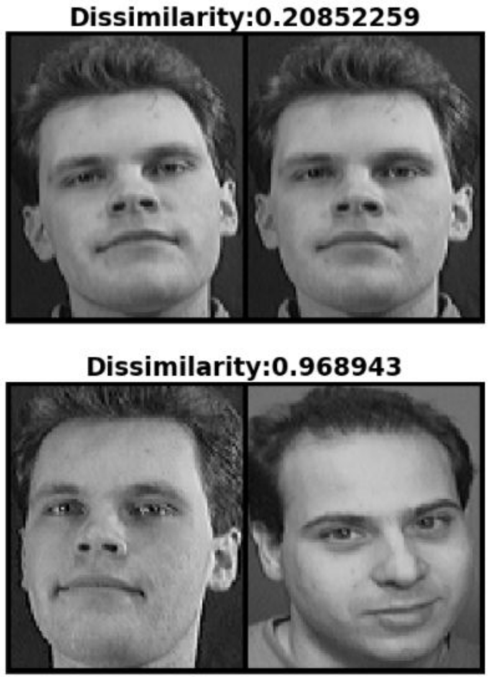
The training loss (*Binary Cross Entropy Loss*) reduces with every iteration.

It can be clearly seen that the model converges as iterations increase.

Results and Discussion

Considering that our implementation has achieved an accuracy that is very close to what the authors outlined, we can expect that a modified version of this model will work well for a number of other applications, not necessarily limited to one-shot learning. These include:

- Video surveillance (pedestrian/vehicle tracking)
- Face recognition for security systems



Results from a face-recognition system built using a Siamese NN. The model outputs the dissimilarity metric.

Challenges in Implementation

Following challenges were encountered while implementing the model:

- The evaluation task specified in the paper was quite complex and replicating that to evaluate our model was a major challenge.
- The paper also mentions that in the sub-networks, each convolutional layer uses different learning rates. However, it is not practical to implement this and we instead used a global learning rate for each layer. Still, the best accuracy achieved (91.5%) was very close to what the authors were able to achieve (92%).
- There was no mention of an appropriate learning rate to use so we had to figure out a suitable rate using trial and error.

Challenges in Implementation

(continued from previous slide)

- The paper highlights how their model beats other deep learning based methods used for the same task. However, we were not able to verify the same with our implementation due to time-constraints.

Future Scope

The paper primarily highlights a verification task for very simple, single-channel images. However, the same model can be extended to handle images with multiple channels (RGB) and to actually classify these images.

Further, the dataset has been augmented using global affine transformations. Drawing from the approach highlighted in the HBPL paper, we could make use of local affine transforms on the individual strokes of the image and train the network with these. This will ensure that the network generalises well to variations.

If tuned perfectly, the model can be used for developing state-of-the-art face recognition and object detection systems. Such systems find widespread use in security checks, e-commerce and logistics industries etc.

Learning Outcomes

There are a number of learning outcomes from this project, both at conceptual and implementation levels:

- Reading the research paper allowed us to develop a familiarity of numerous concepts such as Siamese networks, Convolutional Neural Networks, Training on large datasets, etc.

- We learnt how to translate theoretical concepts learnt from reading a research paper into working code, which further reinforced the concepts and improved our knowledge of the deep learning tech stack that we used, primarily PyTorch.

- One of the main problems tackled by the paper was that of implementing machine learning systems when the data available is limited. We were able to conceptually and practically understand how to overcome such a challenge. Apart from the system itself being one-shot, we were also able to augment the dataset using affine distortions, and thus we understood yet another dimension of possible solutions to the problem of a lack of an expansive dataset.

Learning Outcomes

(continued from previous slide)

- The research community uses specific formats and techniques while publishing their work. By reading a paper directly we were able to understand the importance of such a medium, and how we (as possible future researchers) can use these tools to publish our scientific work in an effective way.

- We learnt how to collaborate in a remote environment, and work effectively as a team, without compromising on the quality of work undertaken. Despite challenges we were able to deliver a working machine learning program, trained and tested, with an accuracy extremely close to that in the paper.

Thank You.