UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL MASTER THESIS

MASTER'S DEGREE IN TELECOMMUNICATIONS ENGINEERING

Semi-Supervised Learning for Training CNNs with Few Data

Author:
Víctor Garcia Satorras

Supervisors:
Joan Bruna Estrach

CILVR Lab
Center for Data Science, NYU

October 13, 2017

Universitat Politècnica de Catalunya

Abstract

Courant Intitute of Mathematical Sciences
Center for Data Science, NYU

Semi-Supervised Learning for Training CNNs with Few Data

by Víctor Garcia Satorras

Although Deep Learning has successfully been applied to many fields, it relies on large amounts of data. In this work we focus on two different research lines within the context of image classification that try to deal with this problem. *a)* The first part of the project is focused on Active Learning (AL), which is an extensive field within Machine Learning that tries to reduce the amount of labeling work by interactively querying the most informative samples from a large dataset. Most of the AL literature is based on uncertainty sampling methods which do not perform so well when applied to neural networks. In this project we present a density estimation approach for Active Learning that overcomes some of the sampling limitations related to the uncertainty-based methods. *b)* The second part of the project is focused on a very recent field within deep learning called one-shot learning, which aims to correctly classify samples by just seeing one or few training samples from each class. In this work we present a simple non-linear learnable metric for one-shot learning that overcomes most of the state of the art results obtained with simple methods and is competitive in terms of accuracy to more complex ones. We also present a meta-learner architecture based on Graph Neural Networks for one-shot learning.

Contents

Chapter 1

Introduction

Deep Learning has become surprisingly good approximating functions that map from inputs to outputs, but it usually relies on large amounts of data. The high costs for labeling such amounts of data, is giving more importance to the unsupervised and semi-supervised fields. In this project we focus on two important domains related to semi-supervised learning:

Active Learning

The first part of the project focuses on Active Learning (AL), which is based on the idea that unlabeled data is easy to get, but labels are expensive. Therefore, AL aims to find the optimal way to select which samples to label in order to obtain the maximum accuracy for a certain task. AL is an extensive field with many years of contributions and a large literature, but regarding to Convolutional Neural Networks most of the classic algorithms do not perform so well. In this project we propose an AL criterion that instead of relying on uncertainty measures, it is based on density estimation measures. The purpose of our algorithm is to wisely choose a subset of samples from a large dataset that better represents the dataset itself. Our assumption is that if the chosen subset and the large dataset are as similar as possible, a classifier trained with the subset will also be similar to the one trained with the whole set.

We are formulating this algorithm and evaluating it with other criteria over different scenarios.

One-shot Learning

Despite recent advances in deep learning, models lack the ability to generalize on new conditions not present in the training data. Usually, image classification requires hundreds or even thousands of images per class. One-shot learning introduced by [8] aims to solve this problem by learning a classifier from just one or few samples.
Learning from one or few samples from scratch and no prior information, is almost an impossible task. In a typical one-shot learning scenario we can distinguish three steps. 1) Representation Learning, model of the world, 2) Learning from the shot (one or few labels) 3) Classifying a given image into its corresponding class from the shot. One shot learning has gained a lot of popularity during the last year, most of the methods consist on learning a representation and then classifying using the given shot at inference time. Our two main contributions of this work are the following:

- The first one is a learnable non-linear metric. Metric learning is an important topic inside one-shot learning. Building a representation model of the world before solving a one-shot task is crucial.

- The second contribution is the use of Graph Neural Networks. Having a global scope of the full one-shot subset before making the classification decision improves the accuracy in some experiments.

We evaluate the two commented apporaches using Omniglot and Mini-Imagenet datasets comparing it to the state of the art results.

Chapter 2

Introduction to Machine Learning

2.1 Machine Learning Basics

Machine learning is a broad field of computer science that provides systems the ability to learn from data without being explicitly programmed.

2.1.1 Types of learning

Machine learning algorithms can be categorized in two main categories based on the nature of the feedback received during the learning process.

- **Supervised Learning**: The algorithm receives a dataset in the format (input, output) pairs, for every input experience the desired output is given by a "teacher", the goal is to learn an algorithm that generalizes for future examples.
- **Unsupervised Learning**: The algorithm experiences a dataset and learns useful properties of the structure, for example, hidden patterns in the data. No labels are provided for this type of learning.

Semi-supervised learning plays an important role in our work, it combines small amounts of labeled data with large amounts of unlabeled one. This strategy can produce significant improvements compared to supervised algorithms when few labeled samples is available.

Role Interchangeability

Roughly speaking, unsupervised learning tries to approximate a distribution of samples $p(\mathbf{x})$ while supervised methods try to estimate a label y from a sample $\mathbf{x}$ 2.1: .

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')} \tag{2.1}$$

Regarding supervised and unsupervised algorithms, the line between them is blurry and not formally defined. For example, given an array of features $\mathbf{x} = \{x_1, x_2, ..., x_K\}$, it can be splitted in K supervised tasks where each feature x_i is estimated from the rest of the features $\mathbf{x} - \{x_i\}$. The core idea is that "fake" supervised tasks can be created to learn in unsupervised problems. One common example is *word2vec*, the dataset can be a large stream of words or sentences, then given a word from the dataset, the algorithm must predict the probability of the nearby words. This simple task can extract powerful patterns from the data.

Alternatively, a supervised learning problem $p(\mathbf{y}|\mathbf{x})$ can be handled as an unsupervised one $p(\mathbf{x}, \mathbf{y})$.

2.1.2 Supervised Training

Given a dataset of N training samples $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ samples from a P distribution, where each $\mathbf{x}_i$ is a feature vector and y_i its ground truth label, we want to find a function $f(\mathbf{x}_i)$ that learns to map samples from the input distribution $\mathbf{X}$ to the output $\mathbf{Y}$, $f : \mathbf{X} \to \mathbf{Y}$.
To measure how good the function f fits the training data, it is possible to assign a Risk value to the function $\hat{y} = f(x)$. The Risk function is defined as the expectation of individual Losses through the training set:

$$R(f) = \frac{1}{N} \sum_{(\mathbf{x},y)\in D} \mathcal{L}(y, f(\mathbf{x})) \tag{2.2}$$

The target function f will be such that minimizes the Risk function:

$$\min_{f\in\mathcal{F}} R(f) \tag{2.3}$$

Loss function

The loss function $\mathcal{L}(y, \hat{y}) \in \mathbb{R}$ is a measure of the error between the ground truth label y and the prediction from our model $\hat{y} = f(\mathbf{x})$. This measure must be carefully chosen, for example, if our function $f(\mathbf{x})$ is approximating a conditional probability distribution $P(y|\mathbf{x})$, the negative log-likelihood is commonly used: $\mathcal{L}(y, \hat{y}) = -log(P(y|\mathbf{x}))$.
Other Loss functions are used for other scenarios, for example, for regression approximations the Mean Square Error is commonly used.

Learning f: X → Y

We already explained how to evaluate the fitness of a function $f(x)$ on the training set. In this section we explain how to fit the function $f(x)$ into the training set, in other words, how we learn the function $f(x)$. In the section 2.1.3 we will explain the generalization paradigm of predictive models on new and unseen data.

Different approaches can be used for approximating predictive models: Decision trees, Rule Learning, Neural Networks, Regression models... In this introduction we just focus on the differentiable models, i.e. neural networks, that can be trained by gradient descent optimization.

We parametrize the function $y = f_\theta(x)$ with parameters θ. The function $f_\theta(x)$ must be composed by differentiable operations. Then using Gradient Descent optimization we can reduce the loss $\mathcal{L}(y, f_\theta(x))$ by substracting the gradient of the loss with respect to the θ parameters.

$$\theta_{t+1} := \theta_t - \alpha_t \sum_{(\mathbf{x},y)} \nabla_{\theta_t} \mathcal{L}(y, f_{\theta_t}(\mathbf{x})) \tag{2.4}$$

In the past, gradient descent has been seen as slow or unreliable. Nowadays, we know that with the enough amount of data, very large non-linear models can be optimized by gradient descent. Stochastic Gradient Descent has significant benefits, the computation cost does not increase with the dataset size, although the number of updates may be larger for large datasets, it also implies a closer convergence to

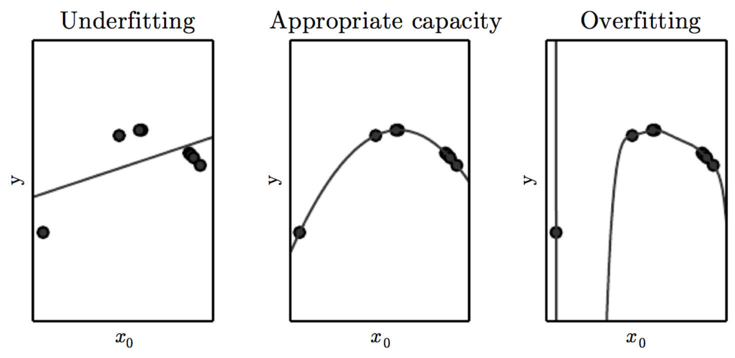


FIGURE 2.1: Left image: High bias algorithm is under-fitting the data. Middle image: Appopriate trade-off between bias-variance. Right image: High variance algorithm is over-fitting the data. *Image from*[10]

the optimum solution. Once reached the best convergence point, increasing the size of the dataset will not increase the time to achieve the same convergence point.

2.1.3 The Bias Variance trade-off

The core challenge in machine learning is performing well on new and unseen data. We explained how to optimize a model $f_\theta : X \rightarrow Y$ on a training set, but there is a risk of *overfitting* or memorizing this training set, and a good training loss does not guarantee good performance on a test set of unseen examples

The difference between machine learning and just memorizing a training set is the capacity to generalize on unseen examples. We introduce two key concepts for understanding this generalization.

- **Variance**: It is the sensitivity to small fluctuations in the training set. High variance algorithms can extract more fluctuations from the training set, but it can also lead to model undesired noise, not present in the test set. Memorizing the training set with a lack of generalization to new samples is called (**Overfitting**)
- **Bias**: High bias algorithms reduce the gap between the training and test losses, but it can also lead to ignore relevant patterns from the data (**Underfitting**).

At figure 2.1 three different cases are represented. From left to right we see a high bias algorithm which under-fits the data, in the middle we see an appropiate trade-off between variance and bias, the right image shows a high variance algorithm over-fitting the data.
At figure 2.2 the curve error vs capacity is plotted. Notice as capacity increases, the training loss is always reduced, since the algorithm becomes more sensitive to the data. The generalization error decreases until the optimal capacity point, and then increases augmenting the generalization gap between training a test set.

2.2 Deep Learning

Many artificial intelligence tasks have been solved by manually extracting specific set of features for each task, and classifying these features with a simple machine

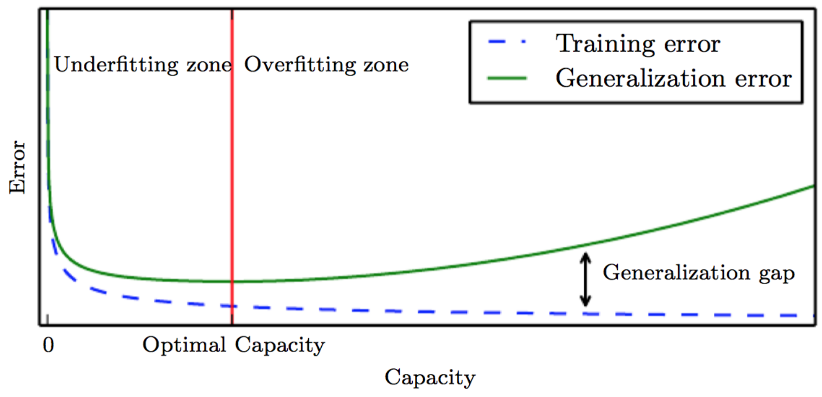


FIGURE 2.2: Left image: Typical capacity trade-off curve. *Image from* [10]

learning algorithm. Manually extracting the right features requires a deep understanding of the task, and it can take days, months or even years and a huge community of researchers.

In order to recognize pedestrians in a street image, we need to hierarchically go from the raw pixel description to the abstract concept of a person. Directly applying a machine learning classification algorithms (i.e. logistic regression) on the raw pixels would produce extremely poor results. Roughly speaking, our image representation should hierarchically go, from pixels $\rightarrow$ edges $\rightarrow$ contours $\rightarrow$ body parts $\rightarrow$ person. For decades these representations have been handcrafted, (i.e Histogram Of Gradients, Contour algorithms, Deformable Parts Model...). Finding this abstract representation space where the desired classes are linearly separable is non trivial.

Representations Learning is a subfield of machine learning that not only tries to learn the mapping from a representation to the desired output, it also tries to learn the representation itself from the raw data. A computer may need just some minutes to extract a good representations from the statistics of the data, while handcrafting descriptors may take a lot of time and work. In spite of it, a lot of variations can be found in the data (i.e. for images: orientation, illumintation, backgrounds...) which makes difficult to disentangle good representations.

Deep Learning solves the central problem of representation learning by hierarchically building higher level feature representations from a combination of lower level ones. Deep learning uses a cascade of many layers that constitute a deep neural network, each layer creates a new feature representation from a non-linear combination of the previous ones. Deep neural networks can be trained end-to-end optimizing the classification Loss by Gradient Descent, the loss is back-propagated through the layers by using the *back-propagation algorithm* which is based on the chain rule. An example of a hierarchy feature representation is shown at 2.3

2.2.1 Historical context of deep learning

Although Deep Learning is known as a new techonology, it can be traced back to decades ago. Deep learning was first inspired as a computer model of neural networks from the human brain. We list some of the milestones in the development of

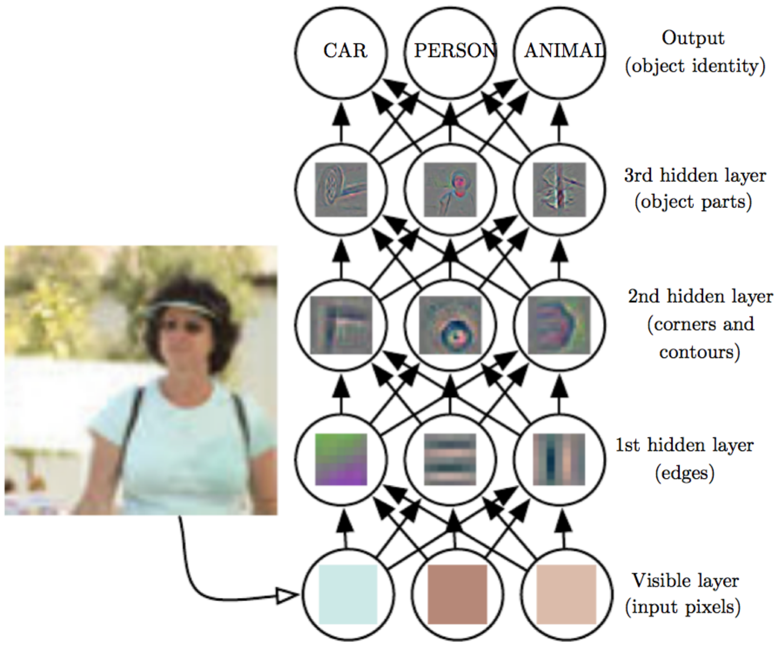


FIGURE 2.3: Hierarchy of features of a Deep Neural Network. *Image from* [10]

neural networks though history:

- **1943**: Warren McCulloch and Walter Pitts created a mathematical model called threshold logic to imitate the behaviour of the human brain. The weights of the model were manually adjusted and non-learnable.
- **1957**: Rosenblatt [31] presented the **Perceptron**, it is a linear combination of features followed by a non-linearity. In its continuation, the ADALINE variation [31] was able to train the weights by gradient descent.
- **1969**: Since the perceptron is basically a linear combination of features followed by a non-linearity, it is unable to learn non-linear functions, even basic functions as XOR. This limiation was commented by Minsky and Papert [26], after this, there was a break in this line of research that took place during the following years.
- **1986**: The **Multilayer Perceptron** was presented by Rumelhart, Hinton, and Williams [33]. It was overcoming the limiations of the perceptron, being able to approximate more complex functions by stacking non-linear layers. To update the parameters of lower layers the **backpropagation** rule was introduced.
- **1989** The backpropagation lead to some early success which is the case of the **Convolutional Neural Nets** presented by LeCun et al.[21]
- **1995**: Support Vector Machines **SVM** were introduced by Cortes and Vapnik [3], becoming the main trending choice. Neural Networks still required a too much data and computational resources for that time.
- **2006**: Deep Learning rises to fame after *Hinton and Salakhutdinov* published the paper [12].

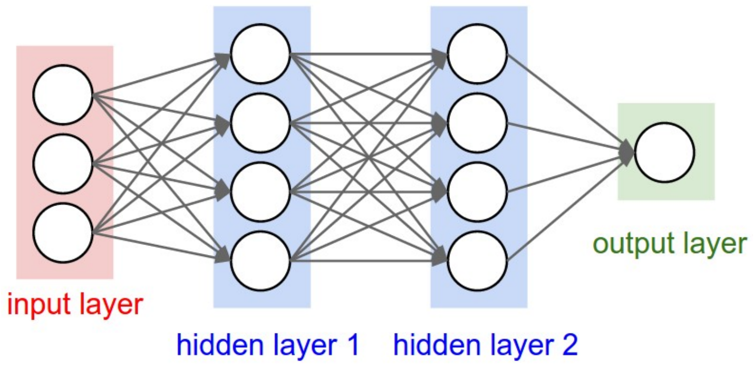


FIGURE 2.4: Multilayer Perceptron. *Image extracted from http://cs231n.github.io/convolutional-networks/*

The breakthrough In 2009 Fei-Fei Li's group from Stanford publihed a 12 milion images dataset [4] Imagenet. This database was used for the Large Scale Visual Recognition Challenge in **2012**. Geoffrey Hinton and Alex Krizhevsky won the LSVRC challenge by training a deep neural network on a GPU reducing by almost half the current state of the art error.

2.2.2 Multilayer Perceptron

The Multilayer Perceptron (MLP) [33], also known as feed-forward neural networks was the first type of artificial neural network devised. The goal of neural networks is to approximate any function f^*. MLP consists of several perceptrons stacked in a layered structure. As we commented before, each a Perceptron is a linear operation followed by a non-linear activation.

$$P(\mathbf{x}) = \sigma \left(\sum_{i=1}^{n} w_i x_i + b \right) \tag{2.5}$$

Where σ is a non-linear operation, i.e sigmoid operation.
The MLP consists of one Input Layer, one or more Hidden Layers, and an Output layer 2.4. Each node in one layer connects with a certain weight w_{ij} to every node in the following layer. In case of no hidden layers, the MLP becomes a Perceptron.
MLP are universal function approximators, it has been shown, that a feed-forward network of one hidden layer with the enough amount of nodes is able to represent any arbitrary function $f(\mathbf{x})$, although it can fail to learn and generalize.
The weights w_{ij} of a neural network are trained via the backpropagation algorithm.

Back-propagation

Back-propagation is a supervised algorithm used to train artificial neural networks. The algorithm is based in two cycles. **a)** An input is propagated through the network storing all intermediate activations, then the error is computed from the output and the ground truth supervision. **b)** Partial derivatives of the error are computed and back-propagated through the network by using the chain rule. Once the gradients are computed, the network weights are updated by using an optimization algorithm typically Stochastic Gradient Descent.

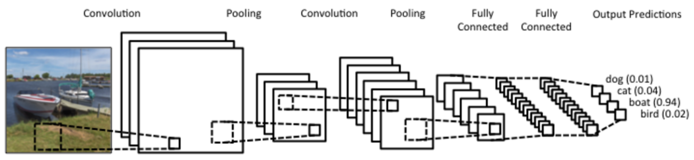


FIGURE 2.5: Convolutional Neural Network. *Image extracted from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/*

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a sub-class of feed-forward neural networks. They also consist of nodes/neurons that receive an input, perform a dot poduct and apply a non-linearity. The main difference of CNNs, is that they explicitly assume some spatial stationarity on the data, typically on images. This assumption allows to add some restrictions on the architecture on the network.
CNN are simply neural networks that instead of using fully-connected layers they use convolutional layers. Based on [10] we distinguish two main features of CNN:

1. **Sparse connectivity**: When dealing with data like images, it is impractical to connect all the output nodes of a layer to all the input nodes. Instead we connect each output neuron, to only a small region of the input, this is accomplished by making a kernel, in the dimensions weight×heigth, smaller than the input dimensions of the image.

2. **Parameter sharing**: As a consequence of the sparse connectivity, and the spatial stationarity of images, we don't need to learn new filter values for every location on the image, instead of that, we can use the same filter parameters for every location along the width×height axis of the image. This naturally introduce the concept of 2-dimensional convolutions where at every layer the local weights are convolved though both axis of the image.

Architecture example

An example of a CNN is shown at image 2.5. We highlight the following parts:

- *Input*: The input is an image of dimensionality $width \times height \times num\_channels$. The number of channels is 3 for RGB images.

- The *Output Predictions* is a vector $P(y|\mathbf{x}) \in \mathbb{R}^C$ that represent the class probability distribution over C classes.

- The *Fully connected layer* is the already commented layer from MLP where each output node is densely connected to all the inputs.

- The *Convolution layer* is the locally connected layer where each output node is connected to a sub-region of the inputs. 2.5.

- *Pooling* is a downsampling operation along the dimensions $width \times height$.

- *A*lthough it is not represented at Figure 2.5, every fully-connected or convolution layer is followed by a non-linear *activation*.

2.2.4 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of unsupervised machine learning algorithms introduced by *Goodfellow et al.* [11] where two neural networks compete with each other in a zero sum-up game.The framework is composed by a generative network G and a discriminator network D. D tries to discriminate whether a given sample comes from the training data distribution or from G. G will try to fool the discriminator generating samples that are similar to the training data distribution.

Formalization: A prior noise distribution $p_z(z)$ is defined and it is mapped by a function $G(z, \theta_g)$ to the data space, where G is a differentiable neural network. The discriminator network $D(x, \theta_d)$ predicts the probability that a sample x comes from the real world. The discriminator is trained to maximize the probability of real x samples and minimize the probability of fake $G(z)$ samples that come from the generator. Simultaneously, the generator is trained in order to fool the discriminator minimizing $log(1 - D(G(z)))$. The final equation of the minimax game is:

$$\underset{G}{min}\, \underset{D}{max}\, V(G, D) = \mathbb{E}_{x \sim p_{data}(\mathbf{x})}[logD(\mathbf{x})] + E_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \tag{2.6}$$

GANs can build powerful representations of the data in a completely unsupervised way by just discriminating fake vs real samples, therefore they are a powerful unsupervised learning tool. GANs can also parametrize the complicated surface of the data distribution and generate very realistic samples from it, therefore they are also very powerful generative model.

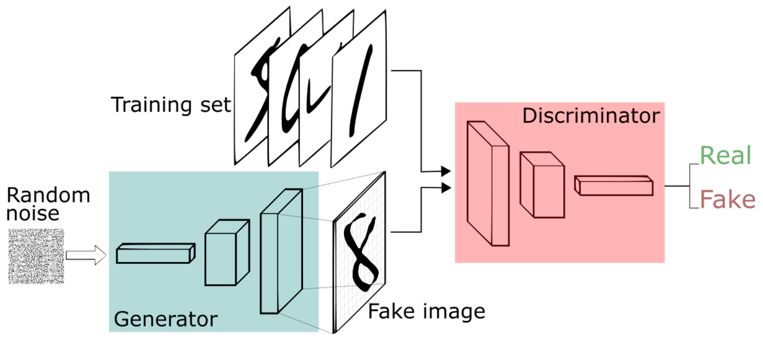


FIGURE 2.6: Generative Adversarial Network. *Image extracted from https://sthalles.github.io/intro-to-gans/*

2.2.5 Graph Neural Networks

Graph Neural Networks (GNN) introduced in [35] and further simplified by [5][24][42] extend the domain of neural networks on graph structured data. The aim of GNN is to approximate a function which input is a graph $G = (V, E)$:

- V (*Nodes*): is a vector of features for every node of the graph. It is represented in a matrix shape $V \in \mathbb{R}^{N \times d}$. N is the number of nodes, d is the number of features per node.
- E (*Adjacent Matrix*): It is a representation of the graph structure in a matrix form $E \in \mathbb{R}^{N \times N}$.

Given the input graph $G = (V, E)$ we consider different intrinsic local operators generated from E that act locally on the signal V. For example, a CNN 2.2.3 can be interpreted as a particularization of a Graph Neural Network were the adjacent matrix E is a binary matrix that defines the adjacent pixels of the image, and the operators are the kernel convolutions along the image dimensions.

In common Graph Neural Network architectures we consider two operators:

- *Degree operator* $D = (d_{ij}) \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$ is a diagonal matrix that contains information about the number of edges attached to each vertex. The operation is a matrix multiplication $D : V \to DV$ where $(DV)_i := deg(i) \cdot V_i$

$$d_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

- *Adjacency operator* $A = (a_{ij}) \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$ contains information about the edges of each node. The operation is $A : V \to A(V)$ where $(AF)_i := \sum_{j\sim i} F_j$ with $i \sim j$ iff $(i, j) \in E$. It is possible to aggregate local information at different scales by adding 2^J powers of A, $\mathcal{A} = \{A, A_1, ..., A_J\}$ or different local operators.

$$a_{i,j} := \begin{cases} 1 & \text{if } i \neq j \text{ and } i, j \text{ connected} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Similarly to CNN, a layer is composed by a local linear operation on each node/vertex V_i, and a non-linear activation is applied after each operation. Layers can be stacked giving rise to a Graph Neural Network. Same as in common NN, a loss is computed and the output is optimized by SGD.

Chapter 3

Active Learning

Active learning is the procedure in which an algorithm is able to interactively query a user/oracle for information in order to maximally perform on a task. This technique is commonly used in contexts where unlabeled data is abundant and asking for labels is expensive, then the algorithm must wisely choose which queries should be asked in order to maximize the accuracy of the target task.
In this project we are focusing in active learning for image classification. Given a large amount of unlabeled images, and the ability to query for the labels of a small subset of them, our algorithm must learn the best classification algorithm by wisely selecting this subset of images to label. In other words, instead of asking a human to label the full dataset of images, we will ask to label just a subset of the dataset, and we want to choose this subset such that we get the best possible classifier.

3.1 Problem definition

We formalize the problem with the following notation:
Given a large set of unlabeled samples $T_U = \{x_i\}_{i=1..N}$, a classification network $P(\hat{y}_i|x_i, \theta) = f_\theta(x_i)$, an oracle that returns the ground truth label for any sample $y_i = \phi(x_i)$. We want to find a subset $T_\Gamma \in T_U$ that maximizes the accuracy of the classifier $f_\theta(\cdot)$ when trained with $\{T_\Gamma, \phi(T_\Gamma)\}$.
The common setup of an iterative Active Learning algorithm for image classification is the following [16]:

1. The algorithm starts with an empty or small subset of labeled samples $T_{\Gamma_{t=0}}$
2. A classifier $f_{\theta_t}(\cdot)$ is trained using the current data $\{T_{\Gamma_t}, \phi(T_{\Gamma_t})\}$. (The training can also be semi-supervised if we include the unlabeled set $\{T_{\Gamma_t}, \phi(T_{\Gamma_t}), T_U\}$).
3. A query selection method $\mathcal{M}$ chooses a new sample(s) x to be labeled where $x \in \{T_U \backslash T_{\Gamma_t}\}$
4. The chosen sample is added to the subset $T_{\Gamma_{t+1}} = \{T_{\Gamma_t} \cup x\}$
5. Repeat from point 2 to 5 until we reach the desired amount of samples or accuracy.

Analyzing the commented procedure, the Active Learning algorithm is reduced to the query selection method $\mathcal{M}$ from point 3. The procedure now is unfolded, and it can be interpreted as a Plug and Play algorithm, where we can add different classifier strategies *point 2* (i.e supervised, semi-supervised classification) and different active learning methods $\mathcal{M}$ *point 3* (i.e Uncertainty sampling, expected error reduction...)
In this work we compare four different methods $\mathcal{M}$:

1. *Uncertainty Sample*: At each iteration we select the most uncertain sample.
2. *Similarity of Distributions*: At each iteration we select a sample that maximizes a similarity metric between $T_{\Gamma_{t+1}}$ and T_U distributions. This method is providing the strongest results.
3. *Best-Greedy*: The sample that maximally reduces the loss is chosen. We show that even the ground truth best-greedy sample performs worse than the *Similarity of Distributions* method.
4. *Random*: A random sample is chosen at each iteration.

3.2 Related work

Active learning has been extensively studied in the last years and different sampling strategies have been developed. Despite of the wide literature, some of these methods are not performing so well when working with high dimensional data and deep CNN.
Uncertainty Sample is a simple method and has demonstrated to properly work on a wide variety of tasks [43] [13] [37] [22], but applied to image classification with large datasets it is pronce to ask for outliers which are not very informative and in some cases it can perform even worse than random. In [23] they combine this method with a information density measure to overcome that limitation.
Information density framework focuses on the input space rather than individual samples, then it is less prone to ask for outliers. It is based on the idea that informative samples should not only be uncertain, but also representative [46] [1] [36]. This concept was introduced by [38] and it is highly related to the method we are presenting in this work.
Expected Error reduction, this decision aims to predict how much the generalization error is expected to reduce when adding new samples to the labeled subset $\{T_\Gamma \cup x\}$ [14] [37] [32].
Learning Active Learning: Some recent work is focusing on how to learn active learning, using Reinforcement Learning [7] or training a regressor to predict the Expected Loss Reduction [16]. These methods must learn over some data and generalize on new datasets.
Although active learning encompasses a lot of works, to the best of our knowledge, we only found three papers related to active learning for image classification using CNN. *Wang et al.*[45] whose method queries low confidence labels, *Stanitsas et al.*[41] presents a comparison over active learning methods for a medical dataset, and *Sener and Savarese*[36] concurrently to our work, proposed a similar geometrical approach to the Similarity of Distributions method that we are presenting.

3.3 Active Learning methods

In this section we introduce the different Active Learning criteria that we are analyzing in our experiments.

3.3.1 Uncertainty Sampling

This is probably the most simple and one of the most famous Active Learning criterion. It consists in querying for the labels which the algorithm is most uncertain of. For example, given an unlabeled dataset T_U and a classification probability $P(y|x, \theta)$ which we approximate with our classification network $y = f_\theta(x)$ s.t. $y \in \mathbb{R}^{num\_classes}$, we would query the label from the most uncertain $x \in T_U$:

$$\hat{x} = \underset{x \in T_U}{\text{argmax}}[H(P(y|x, \theta))] \tag{3.1}$$

The function $H(\cdot)$ from equation 3.1 represents the concept of entropy introduced by *Shannon* [39]. Other uncertainty measures may be used instead of the entropy, for example, given the class label with the highest probability for each x, $\hat{y} = \underset{y}{\text{argmax}}[P(y|x, \theta)]$. We would query for the least confident x:

$$\hat{x} = \underset{x \in T_U}{\text{argmax}}[1 - P(\hat{y}|x, \theta)] \tag{3.2}$$

In our experiments we are using the Shannon's entorpy sampling strategy 3.1 since it is the most popular regarding to uncertainty estimation.

3.3.2 Similarity of Distributions

Uncertainty based methods have their own limitations. They are only based on the individual uncertainty of samples and not in how representative are they in the distribution. It can lead to query very uncertain outliers that are not representative and do not improve the generalization error.

In this section we present an active learning framework for image classification based on finding a subset T_Γ that maximally represents the full dataset distribution T_U. We empirically demonstrate that this strategy gives strongly better results than Uncertainty Sampling when applied to CNN and image classification. The algorithm is based in the following assumption: We want to find a subset T_{Γ^*} from T_U such that minimizes a certain metric distance between both distributions.

Earth Mover's Distance (EMD)

Earth Mover's Distance (EMD) is a measure of distance between two probability distributions over a given metric space. It is also known as the **Wasserstein metric** by the mathematic community. Our objective is then to minimize the distance between the two distributions:

$$T_\Gamma^* = \underset{T_\Gamma}{\text{argmin}}[\text{EMD}(T_U, T_\Gamma)] \tag{3.3}$$

The EMD depends on a metric space, then we need to define a pair-wise distance between points of both distributions. For this we will take the last hidden layer of the classifier neural network $f_\theta(\cdot)$ as the metric space. From now on, we decompose the notation of the classifier $f_\theta(\cdot)$ in two functions $f_\theta(x) = c(e(x))$, where $e(x) \in \mathbb{R}^h$ is the embedding space from which we will compute the pairwise distances between samples: $d_{i,j,\theta} = \text{euclidean}(e_\theta(x_i), e_\theta(x_j))$.

The Earth mover's distance between both sampled distribution will be:

$$\mathrm{EMD}(T_U, T_\Gamma) = \sum_{x_i \in T_U} \sum_{x_j \in T_\Gamma} f_{i,j} d_{i,j} \tag{3.4}$$

Where $f_{i,j}$ is the defined flow $\{0, 1\}$ between two samples $x_i \in T_U, x_j \in T_\Gamma$. For each sample x_i we define the flow $f_{i,j} = 1$ for the closest sample x_j and $f_{i,j} = 0$ for the others, the resulting EMD is:

$$\mathrm{EMD}_2(T_U, T_\Gamma) = \sum_{x_i \in T_U} \min_{x_j \in T_\Gamma} d_{i,j} \tag{3.5}$$

Finally, we just have to choose the T_Γ that minimizes EMD$_2$:

$$T_\Gamma^* = \operatorname*{argmin}_{T_\Gamma}[\mathrm{EMD}_2(T_U, T_\Gamma)] \tag{3.6}$$

Being $T_\Gamma = \{x_1, ..., x_k\}$ a subset of k samples from $T_U = \{x_1, ..., x_N\}$, the optimal solution for 3.7 is a combinatorial problem $\binom{N}{k}$ which becomes computationally impossible for large N datasets. We use a greedy approximation where given T_{Γ_t} we sweep which sample $x^* \in \{T_U - T_{\Gamma t}\}$ will maximally reduce the Earth Mover's Distance equation, then $T_{\Gamma t}$ is updated adding the selected sample $T_{\Gamma t+1} = \{T_{\Gamma t} \cup x^*\}$.

Algorithm

The high-level code for the algorithm is the following:

```
Data: $\{T_U$, K$\}$
Result: $T_\Gamma *$
Initialize: $T_{\Gamma t=0}$, $f_{\theta_{t=0}}(\cdot)$ ;
while num_samples($T_{\Gamma t}$) < K do
    $f_{\theta_t}(x)$ is trained from $\{T_U$, $T_{\Gamma t}$, $f_{\theta_t}(\cdot)\}$;
    for $i \in T_U$ and $j \in T_U$ do
        Distances are computed ;
        $d_{i,j,\theta} = \mathrm{euclidean}(e_{\theta_t}(x_i), e_{\theta_t}(x_j))$
    end
    $x^* = \operatorname*{argmin}_{x \in \{T_U - T_{\Gamma_t}\}} [\mathrm{EMD}_2(T_U, T_{\Gamma_t} \cup x)]$;
    $T_{\Gamma_{t+1}} = T_{\Gamma_t} \cup x^*$ ;
end
```

Algorithm 1: EMD algorithm for Active Learning

Finding a T_Γ that reduces the $EMD(T_\Gamma, T_U)$ is a density estimation criteria that reduces the average distance between points from T_U to the closest points of T_Γ. This is equivalent to algorithms like k-medoids, but k-medoids is more likely to fall in local minimums than our presented method where at each iteration we are trying all possible $x \in T_U - T_\Gamma$.

Computing all the pair-wise distances $D = (d_{ij})$ between elements is of the order $\mathcal{O}(n^2)$. At every iteration the embedding parameters θ_t are recomputed so the matrix of distances D must be also recomputed. It can be slow for large datasets, for that reason we are taking a sub-partition of the dataset at every iteration and selecting x^* from it.

3.3.3 Best Greedy

We've presented two heuristic methods both minimizing a value which is not directly related to the Loss. The Uncertainty Sampling criteria minimizes the entropy of individual samples and the Similarity of Distributions criteria minimizes the metric distance EMD between two distributions.

At this section we introduce a new method to check how well we can perform when at every iteration we choose the sample $x* \in \{T_U - T_\Gamma\}$ that maximally reduces the classification loss $\mathcal{L}(\theta_t)$ of the classifier f_{θ_t} when it is trained with $\{T_\Gamma \cup x^*\}$. When computing the loss $\mathcal{L}_\theta$ we use the training data T_U as a validation set. Notice that we need all the labels from T_U to compute the loss $\mathcal{L}(\theta_t)$, hence it is not an Active Learning criteria but it can be used as a harder baseline than random. It could also be used to train an algorithm that chooses the best sample at every time step, which can be defined as Learning to do Active Learning. The Best-Greedy criterion is the following:

$$x^* = \underset{x \in \{T_U - T_{\Gamma_t}\}}{\text{argmax}} \ (\mathcal{L}(\theta_t) - \mathcal{L}(\theta_t')) \tag{3.7}$$

Where θ_t' are the parameters of the classifier $f_{\theta_t'}$ trained with $\{T_{\Gamma_t} \cup x\}$
In the experiments we will show that, the EMD criteria performs even better than this greedy algorithm.

3.4 Semi-supervised method

Each time a new sample is added to T_{Γ_t}, the parameters θ_t are updated, leading to a new embedding function e_{θ_t}. In order to build a better embedding function for the EMD algorithm, the classifier $f_{\theta_t}(\cdot) = c_{\theta_t}(e_{\theta_t}(\cdot))$ is trained by semi-supervised learning. It means we are also training with the unlabeled data T_U together with the labeled samples T_{Γ_t} and their respective labels $\phi(T_{\Gamma_t})$. We do semi-supervised by using the method from *Salimans et al.*[34], which makes use of Generative Adversarial Networks that we explained at section 2.2.4.

Considering our classification network $f_\theta(\mathbf{x}) = P(y|\mathbf{x}, \theta)$ where $\mathbf{x}$ is the input image and y is a softmax probability distribution over K classes. We can do semi-supervised by simply adding en extra class for the fake samples, increasing the dimension of our classifier from K to $K+1$. The classification cases will be the following:

- **Labeled** T_Γ will be classified among one of the K classes $y \in \{1, ..., K\}$

  $\mathcal{L}_{sup} = -\mathbb{E}_{\mathbf{x},y \sim T_\Gamma, \phi(T_\Gamma)} log[p(y|\mathbf{x}, y < K+1)]$

- **Unlabeled Real** T_U will maximize $y \in \{1, ..., K\}$ and minimize $y = K+1$

  $\mathcal{L}_{uns\_real} = -\mathbb{E}_{\mathbf{x} \sim T_U} log[1 - p(y = K+1|\mathbf{x})]$

- **Unlabeled Fake** T_{Fake}will maximize $y = K+1$ and minimize $y \in \{1, ..., K\}$

  $\mathcal{L}_{uns\_real} = -\mathbb{E}_{\mathbf{x} \sim T_{Fake}} log[p(y = K+1|\mathbf{x})]$

The loss of the discriminator is then:

$$\mathcal{L}_{Discriminator} = \mathcal{L}_{sup} + \mathcal{L}_{uns\_real} + \mathcal{L}_{uns\_real} \tag{3.8}$$

Since the generator is trying to fool the discriminator with the fake data, the Generator Loss is the following:

$$\mathcal{L}_{Generator} = -\mathbb{E}_{\mathbf{x} \sim T_{Fake}} log[1 - p(y = K + 1|\mathbf{x})] \tag{3.9}$$

Our implementation differs in some details from the original one [34]:

1. In the original paper, instead of training the Generator to maximize the Discriminator Loss, they use the technique *(feature matching)* where the objective of the Generator is to minimize the MSE distance between Fake and Real data in an intermediate layer of the discriminator. In our case, we are using the common min, max game of GANs.

2. To avoid instability when training by semi-supervised, we added some noise to the True/Fake labels, it improved the stability considerably when training with few samples.

3. We didn't reduce the number of outputs K+1 to K. In the original paper the last neuron (Fake class) is removed and the value of the K aggregated classes is maximized/minimized for True/Fake images respectively.

3.5 Experiments setup

We present Active Learning benchmarks among {Uncertainty Sampling, Similarity Distribution, Random criteria and Best-Greedy} on MNIST, SVHN and two Dummy datasets.

3.5.1 Datasets

MNIST The MNIST dataset is formed by 10 classes of handwritten digits, it is splitted in 60.000 training examples and 10.000 test examples. Image resolution is 28x28 and there is only one black&white channel. [20]. In our experiments we are using permutation invariant MNIST, which ignores the image structure that CNN exploit and considers every image as a 784-dimensional vector.

SVHN SVHN is a real-world image dataset, it can be seen similar to MNIST (e.g images are also digit numbers) but it is a significantly harder real world problem. The dataset is formed by 10 different classes, 73.257 training samples and 26.032 test samples, images are RGB of 32x32 resolution.

DUMMY-Gaussians

We created an artificial DUMMY dataset for fast testing and prototyping of Active Learning algorithms. It is formed by a set of 9 2-dimensional Gaussians, with diagonal covariance matrix, located in a 3×3 grid, the distance between gaussians is 4σ. We assigned one class to each Gaussian, and the dataset contains 400 training samples and 400 test samples per class.

DUMMY-Uniform

In the same line as *DUMMY-Gaussians*, this dataset is divided into 9 classes, in a 3×3 grid. In this case the data distribution is uniform, the uniform data distributions are adjcanet and they never overlap (In the Guassians dataset, since the distance between them is 4σ, samples can overlap with a small probability).

FIGURE 3.1: MNIST

FIGURE 3.2: SVHN

3.5.2 Architectures

In order to compare our results with the semi-supervised state of the art methods we inspired our architectures on the ones from [34].

SVHN architectures For the experiments related to this dataset, we are using very similar architectures to [34]. The discriminator network is formed by 9 blocks $\{2D\_Convolution \rightarrow BatchNormalization \rightarrow LeakyRelu(0.1)\}$. The architecture is described in more detail at table 3.3. The generator is a 4 layers CNN, same than the Generator from [29].

Block	Output Size	kernel	stride	padding
1	$32 \times 32 \times nf$	3	1	1
2	$32 \times 32 \times nf$	3	1	1
3	$16 \times 16 \times nf$	3	2	1
4	$16 \times 16 \times 2nf$	3	1	1
5	$16 \times 16 \times 2nf$	3	1	1
6	$8 \times 8 \times 2nf$	3	2	1
7	$6 \times 6 \times 2nf$	3	1	0
8	$6 \times 6 \times 2nf$	1	1	0
9	$6 \times 6 \times 2nf$	1	1	0
Global Average Pooling				

FIGURE 3.3: SVHN architecture. Each block is formed by $\{2D\_Convolution \rightarrow BatchNormalization \rightarrow LeakyRelu(0.1)\}$ nf takes values 64 or 96 for SVHN and CIFAR-10 respectively

MNIST architecture For permutation invariant MNIST, the discriminator architecture consists of 5 fully connected layers with {1000, 500, 500, 250, 250} neurons each layer followed by Relu activation. The generator consists of 3 layers of {500, 500, 28×28} followed by Batch Normalization and Relu activation. Gaussian noise $\mathcal{N}(0, 0.5^2)$ was added at the output of each discriminator layer.

DUMMY architecture Architectures are very simple for this dataset. The discriminator consists of 4 fully connected layers with 24 neurons per layer and tanh activation. The generator consists of two fully connected layers of {64, 2} neurons and tanh activation.

3.5.3 Warm/Cold start scenarios

Depending on the number of labeled samples used to initialize classifier $f_{\theta_{t=0}}$ at $t = 0$ we can identify two main settings:

Warm Start In the Warm Start scenario $T_{\Gamma_{t=0}}$ is initialized with few labels and the classifier f_θ is pre-trained with this initialization, in our Warm Start experiments we initialize $T_{\Gamma_{t=0}}$ with 1 label per class for the DUMMY and MNIST datasets, and two labels per class for SVHN.
In most of the real world active learning scenarios a small amount of data is available before querying for new data.

Cold start In the cold start scenario no labels are provided at $t = 0$ and the algorithm starts querying from scratch. Notice the error can be much higher since the Cold start scenario does not guarantee a minimum amount of labels per class.

3.6 Experiments

We compare the {Uncertainty Sampling method, Random querying, and Our Distribution Similarity method}, for all the datasets. We also present a comparison of these methods with the Best-Greedy approach for the Dummy and MNIST datasets. All experiments have been averaged over 5 runs.

3.6.1 Dummy experiments

Dummy-Uniform

The results for the Dummy-Uniform dataset are presented at Figure 3.4 (Warm Start) and at Figure 3.5 (Cold Start). A visual example of the chosen samples is shown at the (4×4)-grid from figure 3.10. For this dataset, *Our Method* performs better than the *Random* case, and even better than the *Best-Greedy* criterion. We want to remember than the *Best-Greedy* criterion is not actually an Active Learning method since all the labels are needed to compute it, but it is a good reference to check how good are we doing with respect to choosing the sample that most improves the accuracy of the dataset on the current iteration.

The *Uncertainty Sampling* is performing even worse than *Random* for both Warm Start and Cold Start. At first sight it may seem incoherent and we can raise two questions. 1) Is the US a bad criterion for this dataset? 2) Is the uncertainty estimation wrong? Looking at the selected points from images B) and C) from Figure 3.10 we can try to guess the answer. At image B) Warm Start, we see that the algorithm is querying samples from regions of maximum uncertainty of the distribution, which are the four points where the four classes intersect. Because of the low dimensionality of the dataset, it can be hard for the classifier to overfit and reduce the uncertainty of this regions even asking a lot of them.

Looking at the Cold Start image C) from Figure 3.10, the US is only querying samples from two of the four regions of maximum uncertainty, this is because the classifier didn't had the chance to explore some of the classes, so it is querying samples from the only regions it 'knows'.

Method	18	27	36	50	100	150
Random	74.2%	78.3%	81.5%	85.8%	90.6%	91.3%
US	73.1%	76.2%	79.9%	80.0%	84.2%	86.5%
Best-Greedy	79.5%	81.9%	87.1%	88.4%	90.6%	91.9%
Our Method	80.5%	85.8%	88.7%	91.5%	93.9%	95.2%

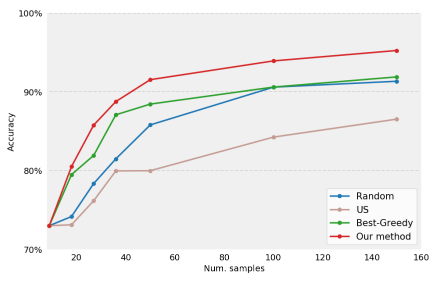


FIGURE 3.4: Dummy-Uniform Warm Start results

Method	9	18	27	36	50	100	150
Random	51.3%	69.1%	76.1%	80.7%	85.1%	87.0%	91.6%
US	46.5%	58.5%	63.6%	64.7%	67.2%	67.8%	69.3%
Our Method	70.9%	81.0%	84.9%	86.1%	89.0%	93.1%	93.2%

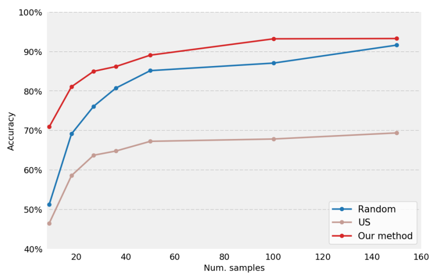


FIGURE 3.5: Dummy-Uniform Cold Start results

Dummy-Gaussian

The behavior for the Dummy-Gaussian dataset represented at Figures 3.6, 3.7 is similar to the Dummy-Uniform. *Our Method* performs better than *Random* and even better than *Best-Greedy* too. The *US* in the Warm Start scenario is still worse than *Random* in most iterations, although the accuracy at the beginning and at the end is much better than with the Dummy-Uniform.

In the Cold Start scenario the algorithm also collapses asking for labels from some of the maximum uncertainty regions and ignoring the unexplored space. This behavior can be visualized at Figure 3.11.

Method	18	27	36	50	100	150
Random	82.5%	85.4%	87.2%	89.7%	91.5%	91.7%
US	85.1%	85.6%	83.2%	81.4%	87.4%	91.2%
Best-Greedy	88.9%	89.6%	89.8%	90.1%	91.6%	91.9%
Our method	90.0%	90.2%	90.8%	91.3%	91.9%	92.6%

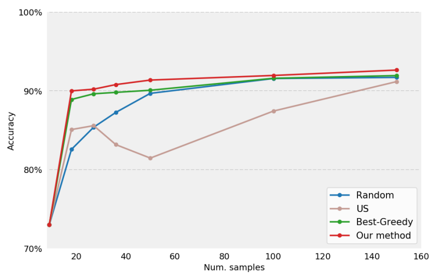


FIGURE 3.6: Dummy-Gaussian Warm Start results

Method	9	18	27	36	50	100	150
Random	55.7%	81.0%	81.9%	83.3%	87.2%	87.5%	91.8%
US	49.7%	58.7%	60.9%	64.1%	63.8%	63.9%	69.6%
Our Method	84.8%	89.8%	89.9%	90.4%	90.8%	92.0%	92.4%

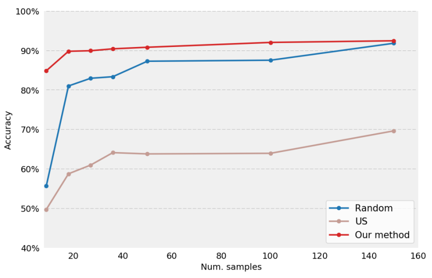


FIGURE 3.7: Dummy-Gaussian Cold Start results

3.6.2 MNIST & SVHN experiments

Although dummy datasets are useful to visualize and prototype algorithms, the behavior of Active Learning methods can vary a lot from low dimensional dummy distributions to high dimensional images. In this section we comment the AL results for MNIST and SVHN datasets, we present a comparison among the methods: {*Random, US, Best-Greedy (only MNIST), Our Method*}. See Figure 3.8 for the MNIST comparison and Figure 3.9 for the SVHN. *Uncertainty Sampling* criteria in large datasets are prompt to ask for outliers, even though, in our experiments, the US results are satisfactory for both MNIST and SVHN datasets, providing greater accuracy than the *Random* strategies. In spite of it, *Our Method* is getting better results than *US*.

Method	20	50	100	200	500
Random	44.9%	63.2%	74.5%	82.7%	89.8%
US	49.8%	64.2%	75.5%	85.6%	92.8%
Best-Greedy	52.1%	69.0%	77.8%	84.0%	90.4%
Our Method	52.2%	67.3%	77.9%	86.8%	93.1%

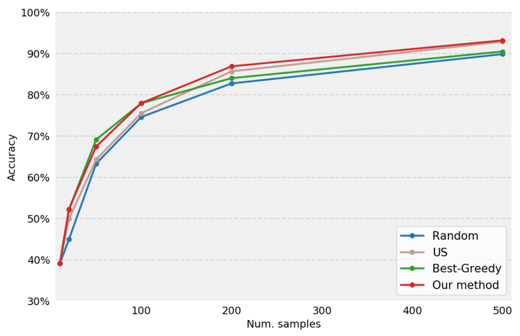


FIGURE 3.8: MNIST results

Method	100	250	500	1000
Random	25.6%	59.8%	77.5%	87.7%
US	26.3%	61.2%	78.5%	87.8%
Our Method	26.2%	64.4%	80.7%	88.8%

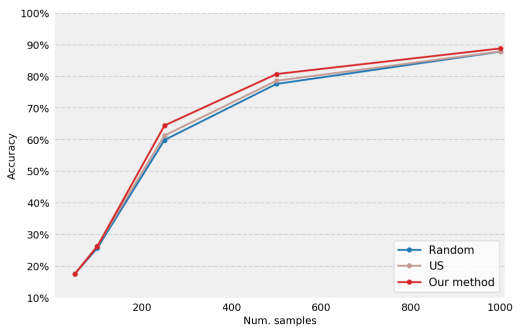
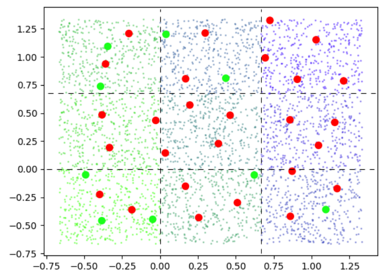
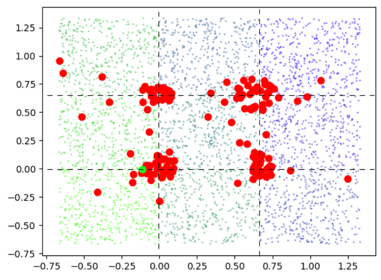


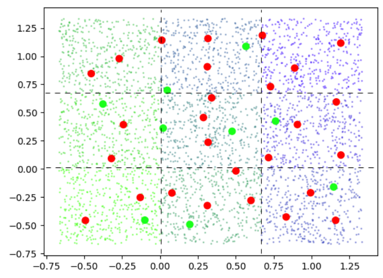
FIGURE 3.9: SVHN results



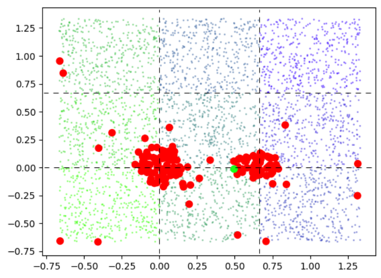
(A) Our Method, Warm start



(B) US method, Warm start



(C) Our method, Cold-start

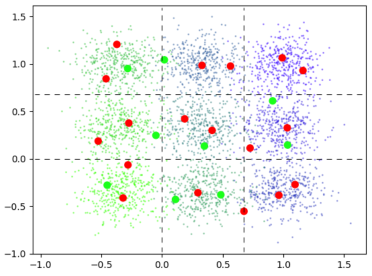


(D) US method, Cold start

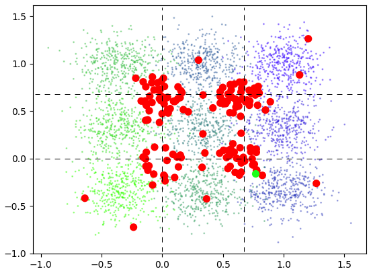
FIGURE 3.10: Sampled labels for DUMMY-Uniform using our method (*Left column*) and Uncertainty Sampling method (*Right column*). Warm start setting (*First row*) and Cold start setting (*Second row*).

3.7 Active Learning Conclusions

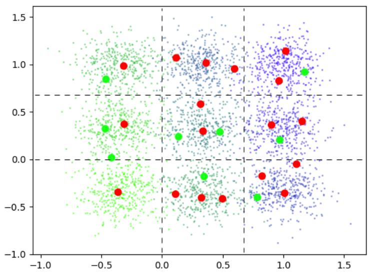
In this work we introduced an Active Learning criterion which intends to find a subset of samples from a large dataset that best approximates the dataset itself. To measure this similarity between the dataset and its subset we have used the Earth Mover's Distance over a metric space defined by the classifier/CNN. We showed with some Dummy and Image datasets how this criterion overcomes some of the limitations of classical Uncertainty Sampling which do not perform as well on Neural Networks.



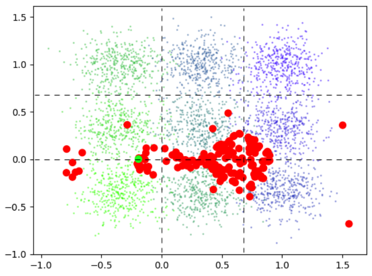
(A) Our method, Warm start



(B) US method, Warm start



(C) Our method, Cold-start



(D) US method, Cold start

FIGURE 3.11: Sampled labels for DUMMY-Gaussian using our method (*Left column*) and Uncertainty Sampling method (*Right column*). Warm start setting (*First row*) and Cold start setting (*Second row*)

Chapter 4

One-shot learning

Most deep learning models require huge amounts of data in order to be trained. Deep neural networks struggle when they are trained with few samples, which is a quite common scenario when there is not enough data for a certain task. The one-shot learning setting aims to correctly classify samples given only one example from each class. This concept can be extended to few-shot learning where an algorithm only has access to few examples from each class.
One common example would be the easiness of humans to recognize an object that has been seen just once. I.e. if a segway is shown to a person for the first time, it is enough for building a mental classification model in order to recognize new segways in the future, and it is not needed to show hundreds of segways in different positions, orientations, colors... as we usually do with neural networks.

4.1 Problem definition

We formalize the problem with the following notation:
Given an unlabeled image $\hat{x}$ and a small subset of *(image, label)* pairs $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ where each label $y_i \in \mathbb{R}^{N_c}$ is codified by a one-hot binary vector of dimensionality N_c, being N_c the number of classes.. We want to find a classifier $\hat{y} = C(S, \hat{x})$ which given the subset S predicts from which of the N_c classes is $\hat{x}$ from.
In figure 4.1, a visual example of one-shot learning for a subset of five different classes is depicted. In this case, we say it is a 1-shot, 5-way problem where "N_c-way" represents the number of classes and "N_s-shot" the number of images per class being $N_c = 5$ and $N_s = 1$ for this particular example.

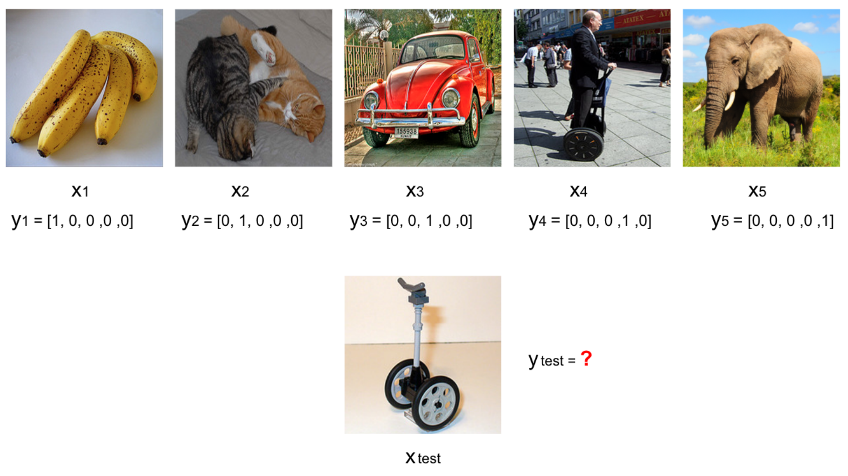


FIGURE 4.1: Visual example of a 1-shot 5-way problem.

We are presenting two **methods**:

1. The first method is a k-nearest neighbors algorithm (k-NN) that makes use of a non-linear learnable metric. *[44]* introduced a end-to-end trainable k-NN for one-shot learning using the cosine similarity metric between image embeddings, *[40]* extended this work providing better results when using the euclidean distance.

   Given the embedding vectors of features from two different images we are training a neural network that non-linearly combines the distance between the individual features in that embedding space. We show that simple architectures that follow this approach can lead to considerable improvements over more complex recent models.

2. Instead of just weighting similarities/distances between pairs of elements as in the k-NN approach, in this second method we want each sample x_i to have knowledge of the full subset S. To fullfill this purpose we use Graph Neural Networks. We will explain this method in more detail at section 2.2.5

4.2 Method 1 | Non-Linear metrics

Following the notation from [44], we can approximate $\hat{y} = C(S, \hat{x})$ as a k-NN in the following way:

$$\hat{y} = \sum_{i=1}^{N} \text{sim}(\hat{x}, x_i) y_i \tag{4.1}$$

Where $\text{sim}(\hat{x}, x_i) \in [0, 1]$ is a measure of similarity between two samples.
The similarity between samples can be decomposed in the following way $\text{sim}(\hat{x}, x_i) = m(f(\hat{x}), f(x_i))$ where:

1. $\mathbf{f}(\cdot)$ is an embedding function that maps from an image x to an embedding vector of low-dimensionality.

2. $\mathbf{m}(\cdot, \cdot)$ is a similarity metric between the embedding vectors $f(x)$. *Vinyals et al.* [44] introduced a end-to-end trainable k-NN for one-shot learning using the cosine similarity metric between image embeddings, *Snell, Swersky, and Zemel* [40] extended this work providing better results when using the euclidean distance. In [25] they train a residual network for approximating the distance function.

   In this work we present a very simple and trainable metric $m(\cdot, \cdot)$ that can lead to considerable improvements over more complex recent models.

4.2.1 Metric Learning

Following the survey [1] we distinguish between two different family metrics:

- Linear metrics: Their expressive power is limited, but they are easier to optimize and usually provide better generalization. They often give rise to convex formulations.

- Non-Linear metrics: They can capture non-linear variations in the data but they are also more susceptible to overfit and can give rise to non-convex formulations.

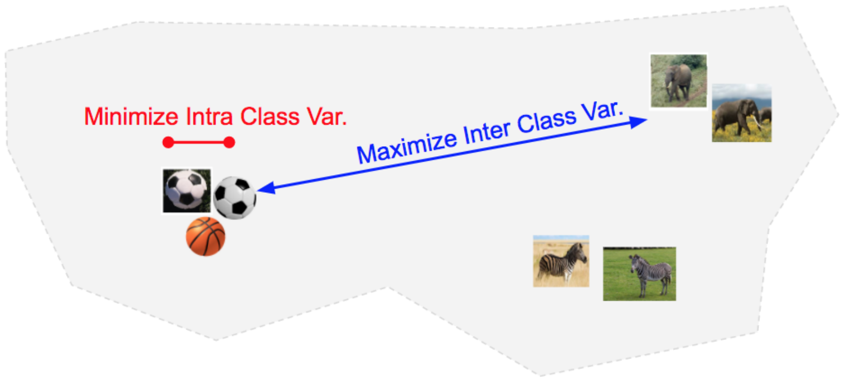


FIGURE 4.2: Intra-Class variance minimization, Inter-class variance maximization.

Motivation of non-linear metrics

Focusing on the simple KNN approach for one-shot learning from equation 4.1. The similarity function sim(x_i, x_j) will output high values when x_i and x_j come from the same class, and low values when they come from different classes. Using a classic similarity metric i.e cosine similarity, the output for two similar x_i, x_j will be close to 1, and for two perpendicular x_i, x_j will be close to 0.

In terms of metric distance, the one-shot training is equivalent to reduce the **Intra-Class Variance** and to increase the **Inter-Class Variance**, as it is depicted in image 4.2. When using a common metric like cosine similarity or euclidean distance for $\mathbf{m}(\cdot,\cdot)$, it will lead to similar embedding representations $\mathbf{f}(\mathbf{x})$ for all the samples that come from the same class. We presuppose that some intra-class variance features from some classes can be useful to discriminate between other classes, and it could be interesting to avoid collapsing them while ignoring it whenever it is convenient. It leads us to think about a non-linear combination of the features, where for example we can have two different vector representations (a, b) from the same class, and still compute a distance equal to 0.

Method

Linear metrics are unable to model non-linear structures that may be found in the data. In order to capture these properties, we are learning a metric from the data that non-linearly combines the distances between the individual features in the embedding space.

Given two embedding vectors $z_1 \in \mathbb{R}^{N_f}$ and $z_2 \in \mathbb{R}^{N_f}$ the L_p, the distance is computed as follows:

$$L_p(z_1, z_2) = ||z_1 - z_2||_p = \left(\sum_{i=1}^{N_f} |z_1(i) - z_2(i)|^p \right)^{\frac{1}{p}} \tag{4.2}$$

In the KNN setting, this distance can be re-scaled, or converted to a similarity by multiplying it with some learned weights plus a bias, in such a way that a Softmax can be added over the different distances between the unlabelled $\hat{x}$ and the labelled $\{x_1, x_2, ..., x_N\}$ to compute the class probabilities.

$$L_{p\_weighted}(z_1, z_2, \theta) = \left(\sum_{i=1}^{N_f} w_\theta(i)^p \cdot |z_1(i) - z_2(i)|^p \right)^{\frac{1}{p}} + b_\theta \tag{4.3}$$

This last equation 4.3 can be interpreted as a weighted sum of the individual feature distances between two vectors. By adding a non-linearity (Softmax or Sigmoid) we can map it to a probability for our one-shot classifier. Notice that 4.3 followed by a non-linearity is equivalent to the Perceptron architecture 2.5 that we explained at the introduction, and we commented the Perceptron limitations in terms of function approximation. In order to learn a distance by a non-linear combination of the features, we only need to stack additional layers to the vector of individual distances $abs(z1 - z2) \in \mathbb{R}^{N_f}$.
Therefore, we present an architecture $m(\cdot,\cdot)$ which is composed of an absolute distance between the feature vector representation of two images followed by two or more fully-connected layers with a non-linearity. In the following line we show an example of $m(\cdot,\cdot)$ in its simplest form when using one fully connected hidden layer and one output layer.

$$m(f(x_i), f(x_j)) = abs(f(x_i) - f(x_j)) \rightarrow fc(k) \rightarrow Relu \rightarrow fc(1) \rightarrow Activation \tag{4.4}$$

This architecture is assuming some prior constraints that directly satisfy some of the properties of a distance/similarity metric. We are briefly commenting this distance properties. By an abuse of notation we consider $m(\cdot,\cdot)$ as a distance metric instead of a similarity metric when listing this distance properties (Notice $m(\cdot,\cdot)$ can be trained to be both things):

1. *Simmetry* $m(a, b) = m(b, a)$: This is fulfilled by construction since the operation $abs(a - b)$ is commutative.

2. *Identity* $m(a, a) = 0$: When the two embedding vectors are equal, all the components of the vector of feature distances $abs(a - b)$ are always 0, this makes this property easily learnable.

3. *Non-negative* $m(a, b) > 0$: This can be accomplished by a positive activation in the last layer. I.e a sigmoid.

The fourth distance property *Triangle inequality* $(m(a, c) < m(a, b) + m(b, c))$ is not necessarily fulfilled by non-linear metrics.

Using this simple architecture we have been able to outperform much more complicated one-shot learning methods.

4.2.2 Learning procedure

Episodic tasks: Given a large dataset D with a large number of classes. We form a few-shot task T_i by randomly selecting N_c classes from D and then sampling N_s examples from each selected class, obtaining a subset S_i of $N_s * N_c$ samples. We also select a support subset B of training/test classification samples from the same classes than S_i.

Following the explanation from section 4.1 in order to solve a task we want to classify a image $x \in B$ into its corresponding class $y_{gt} \in B$. Then our objective function is to reduce the following expected loss with respect to the parametres θ:

$$\hat{\theta} = \underset{\theta}{\text{argmin}}\, \mathbb{E}_{T_i \sim T}\Big[\sum_{(x,y_{gt})\in B} \mathcal{L}(y, C_\theta(S_i, x)) \Big] \tag{4.5}$$

Loss: from equation 4.5 a classification loss $\mathcal{L}$ must be chosen in order to train the algorithm. A basic cross entropy loss works for this purpose. Following this approach, the one-shot learning setup which can be seen as a complicated transfer learning problem, has been simplified to a simple supervised classification problem.

4.3 Method 2 | GNN for contextual information

We presented a method that computes a non-linear pair-wise similarity between $\{\hat{x}\}$, $\{x_{1,..,x_N}\}$ samples, and assigns a probability to each class by weighting these similarities. Notice the distance between samples is blind to the full subset $\{(x_1, y_1), .., (x_N, y_N)\}$, and a strong assumption is done when classifying by means of a weighting sum of pair-wise distances. The first paper in introducing a method that takes into account the full subset S was [44], other contextual approaches followed this work [27] [40]. Some of this works use sequence dependent methods for regulating the

We are presenting a contextual method by using Graph Neural Networks that considers the full subset S. GNNs do not depend on the order of the input data and they are flexible architectures. We think they can be a promising approach for the one-shot learning problem. Graph Neural Networks are briefly introduced at section 2.2.5.

Graph Definition

The first step is to formulate the one-short learning framework into a graph $G(V, E)$ domain:

- *V (nodes)*: Every image from a subset episode $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ and the unlabeled image to classify $\hat{x}$ are encoded into a embedding representation $z = f(x)$, $z \in \mathbb{R}^{1\times r}$ being r the number of features. Inspired on the metric learning approach from our previous method 4.2.1, we compute a vector of feature distances $d_i = abs(\hat{z} - z_i) \in \mathbb{R}^{1\times r}$. Considering every vector d_i as a node of the graph, the matrix of nodes has a shape $V \in \mathbb{R}^{N\times r}$.
- *E (Adjacent matrix)*: As adjacent matrix $E = (e_{ij}) \in \mathbb{R}^{N\times N}$ we use a binary matrix $e_{i,j} \in 0, 1$, if the respective x_i and x_j images from two nodes d_i and d_j come from the same class, then $e_{i,j} = 1$, otherwise $e_{i,j} = 0$.

Local Operators

We are using three local operators, *1) Degree Operator*, D, commented in the introduction 2.2.5, *2) Adjacency Operator*, A, also commented in the introduction. *3)* we are adding a third operator $U = (u_{ij})$

$$u_{i,j} := \begin{cases} 1 & \text{if } a_{ij} = 0 \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The three operators are normalized by row, such that the sum of every row is 1. This third operator broadcasts information from nodes that belong to a different class of the local node.

Coarsening / Pooling

The graph equivalence of *pooling* is *coarsening* [2]. In this operation a α fraction of the nodes are retained. In our network we are performing an Average Coarsening, equivalent to an Average Pooling for CNNs. The one-shot learning graph is organized in N_c disjoint subgraphs of K samples each one, being N_c the number of classes and K the number of samples per class, hence we just need to average the inter-connected nodes from each class, and we get a single node per class, mapping each node to one single neuron and applying the Softmax activation we get the probability distribution $P(y|\hat{x}, S)$. This operation is similar to the average from [40] where they compute the mean of the feature samples before subtracting it to $\hat{x}$, in our case it is applied after subtracting the $\hat{x}$ at the last layer of the GNN. The prototypical behavior of [40] the could also be implemented in terms of GNN using the Laplace operator at the first layer of the GNN.

4.4 Related work

One of the first works in one-shot learning dates back to 2006 by Fei-Fei, Fergus, and Perona [8], using a Bayesian Framework, they assumed that currently learned classes can help to make predictions on new ones when just one or few labels are available. More recentlly, in 2013 [19] presented a Hierarchical Bayesian model that reached human level error.

This last year, a great progress has been done in one-shot learning, a wide varety of papers have been published. Koch, Zemel, and Salakhutdinov [15] presented a deep-learning model based on computing the pair-wise distance between samples using Siamese Networks, then, this learned distance can be used to solve one-shot problems by k-nearest neighbors classification. [44] Presented an end-to-end trainable K-NN using the cosine distance. They also introduced a contextual mechanism using an attention LSTM model that takes into account all the samples of the subset S when computing the pair-wise distance between samples. [40] extended the work from [44], by using euclidean distance instead of cosine which provided significant improvements, they also build a prototype representation of each class for the few-shot learning scenario. Mehrotra and Dukkipati [25] trained a deep residual network together with a generative model to approximate the pair-wise distance between samples.

A new line of meta-learning methods for one-shot learning is rising lately: Ravi and Larochelle [30] introduced a Meta-Learning method where an LSTM updates the weights of classifier for a given episode. Munkhdalai and Yu [28] also presented a Meta-Learning architecture that learns meta-level knowledge across, and it changes its inductive bias via fast parametrization. It is also using an external memory mechanism that helps the generalization. Finn, Abbeel, and Levine [9] are using a model agnostic meta learner based on gradient descent, the goal is to train a classification model such that given a new task, a small amount of gradient steps with few data will be enough to generalize. Lately, Mishra et al. [27] used Temporal Convolutions which is a deep recurrent network that uses dilated convolutions, this method also exploits contextual information from the subset S providing very good results.

In our work we presented a basic approach for learning a similarity metric between pair-wise samples by adding some restrictions to the network architecture, we also presented a contextual method based on Graph Neural Networks which unlike recurrent methods, the output is invariant from the order of the input samples S.

4.5 Experiments

For the one-shot experiments we used the Omniglot dataset presented by *Lake, Salakhutdinov, and Tenenbaum* [18] and mini-Imagenet dataset *Vinyals et al.*[44] which is a small version of ILSVRC-12 [17]. All experiments are based on the N_c-way N_s-shot setting. For all experiments we used the same values N_c and N_s for both training and testing.

4.5.1 Omniglot

Dataset: Omniglot is a dataset of 1623 characters from 50 different alphabets, each character/class has been drawn by 20 different people. Following [44] implementation we split the dataset in 1200 classes for training and the remaining 423 for testing and we increased the training data by random rotations multiples of 90 degrees.

Architecture: Inspired by the *Embedding* architecture from [44], a CNN was used as an embedding function $f(\cdot)$ consisting of four stacked 3x3-convolution layers with 64 filters followed by a Batch-Normalization, a 2x2 max-pooling, and a leaky-relu, resulting in a 1x1x64 embedding. For the *Metric Network* $m(\cdot,\cdot)$ we used a fully-connected neural network of 4 hidden layers each of them followed by a leaky-relu activation, the number of the neurons at each hidden layer is {128, 128, 64, 64} respectively. We didn't use the contextual *GNN* in the Omniglot experiments since we didn't notice significant improvements for this dataset. All models have been trained using Adam optimizer, batch size 40, during 100.000 iterations, initializing the learning rate at 1e-3.

Results: Results are shown at table 4.3. We present three different experiments.

- *Our Euclidean:* is our own implementation for the euclidean distance method which we approximate by removing all the hidden layers from our metric network $m(\cdot,\cdot)$. It is equivalent to the weighted euclidean distance commented at 4.3. This method is already very powerful as it was demonstrated by [40].
- *Our Learnable Metric:* It is the already explained implementation of our algorithm with 4 layers for the embedding network, and 4 hidden layers for the metric network.
- *Our Learnable Metric*:* it is a variation of our model where we add one extra convolutional layer to the embedding network $f(x)$ to compare it with the other methods that are also using 5 layers for the embedding network.

Our Learnable Metric is providing competitive results while remaining very simple in terms of methodology and architecture. The only difference between our method and Matching Networks for this experiment [44] is the trainable distance instead of cosine distance. *Residual Pair-Wise* [25] is also learning a metric from the data using Residual Networks, but since no prior is imposed, the network has to learn the full concept of distance from the data what makes the learning harder.

	5-Way		20-Way	
Model	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Pixels [44]	41.7%	63.2%	26.7%	42.6%
Siamese Net [15]	97.3%	98.4%	88.2%	97.0%
Matching Networks [44]	98.1%	98.9%	93.8%	98.5%
Neural Statistician [6] **	98.1%	99.5%	93.2%	98.1%
Residual Pair-Wise [25] *	-	-	94.8%	-
Prototypic Net. [40]	97.4%	99.3%	95.4%	98.8%
Agnostic Meta-learner [9]	98.7 ±0.4%	99.9 ±0.3%	95.8 ±0.3%	98.9 ±0.2%
Meta Networks [28]*	98.9%	-	97.0%	-
TCML [27]*	98.96% ±0.20%	99.75% ±0.11%	97.64% ±0.30%	99.36% ±0.18%
Our Euclidean	98.6%	99.5%	95.5%	98.3%
Our Learnable Metric	98.8%	99.5%	96.6%	98.5%
Our Learnable Metric*	99.0%	99.6%	97.0%	98.5%

FIGURE 4.3: Omniglot results. All models are using 4 layers with 64 filters per layer for the embedding network $f(x)$ except those marked with a * which are using 5 layers. Those marked with ** are using a completely different architecture.

Our network is giving stronger results for the case of 1-shot than for the few-shot scenario, notice that for the 1-shot 5-Way is giving the best results along with [27] [28] that are using more complex methods like external memory modules [28], Fast/Slow weights [28] or RNN by means of dilated convolutions [28]. We didn't notice significant improvements using our contextual GNN in the Omniglot dataset.

4.5.2 Mini-Imagenet

Dataset: Mini-imagenet is a more challenging dataset for one-shot learning proposed by [44] derived from the original ILSVRC-12 dataset [17]. It consists of 84x84 RGB images from 100 different classes with 600 samples per class. It was created with the purpose of increasing the complexity for one-shot tasks while keeping the simplicity of a light size dataset that still fits in memory on modern machines what makes it suitable for fast prototyping. We used the split proposed by *Ravi and Larochelle* [30] with 64 classes for training, 16 for validation and 20 for test. We used 80 classes for training and 20 for testing in the same way as [44][27].

Architecture: The *Embedding* architecture used for mini-imagenet is extremely simple which is useful for fast prototyping. Similarly to the Omniglot architecture consists of 4 layers: 2 $\times$ {conv_layer, batch-normalization, max_pool$(2, 2)$, leaky relu}, 1 $\times$ {conv_layer, batch-normalization, leaky relu} and 1 $\times$ { fc_layer}. All layers have 64 neurons, and kernel size 3, the first layer has stride 2.
For the *Learnable Metric* we are using exactly the same architecture as in Omniglot experimens.
The *Graph Neural Network* architecture is formed by 4 $\times$ {Graph Conv., Batch Norm., Leaky Relu} and 1 $\times$ {Average Pooling, Graph Conv, Softmax}.

Results: Results on mini-imagenet are shown at the table 4.4. Mini-Imagenet is a much harder task than Omniglot. The GNN that didn't improve the performance on Omniglot improved it on Mini-Imagenet, especially in 5-shot learning where the accuracy increased by $\sim 3\%$.

	5-Way	
Model	**1-shot**	**5-shot**
Matching Networks [44]	43.6%	55.3%
Prototypical Networks [40]	46.61% ±0.78%	65.77% ±0.70%
Model Agnostic Meta-learner [9]	48.70% ±1.84%	63.1% ±0.92%
Meta Networks [28]	49.21% ±0.96	-
Ravi and Larochelle [30]	43.4% ±0.77%	60.2% ±0.71%
TCML [27]	55.71% ±0.99%	68.88% ±0.92%
Our Learnable Metric	48.3% ± 0.46%	61.4% ±0.42%
Our GNN	48.8% ± 0.27%	64.3% ±0.25%

FIGURE 4.4: Mini-Imagenet results

4.6 One-shot Learning Conclusion

One-shot learning has gained a lot of popularity this last year. We presented two main contributions in this work. 1) The learnable non-linear metric, 2) The Graph Neural Network for contextual information.

1. We showed it is possible to get very competitive results with respect to more complex methods by using a non-lineaer combination of features to compute a similarity metric. Although it produced good improvements in the Omniglot dataset, we didn't notice significant changes in Mini-Imagenet.

2. For Mini-Imagenet we introduced a Graph Neural Network as a meta-learner structure that is able to see all the support set $S = \{(x_1, y_1), ..., (x_N, y_N)\}$. It significantly improved the results for the 5-Way 5-Shot case. Graph Neural Networks are very flexible structures and invariant to input permutations which is the case for the subset S, which leads us to think that GNN can be further optimized for one-shot learning tasks.

Chapter 5

Conclusions

Semi-supervised learning is a topic of great interest these days. Larger amounts of data are stored every day and labeling them all is an impossible and expensive task. As a consequence, unsupervised and semi-supervised learning are gaining ground into the field of machine learning.
In this paper we focused on the topics of Active Learning and One-shot Learning for image classification:

Active Learning for image datasets is a very bounded problem, as the span of actions that can be performed by the algorithm is limited by the number of samples. Therefore, the degrees of freedom for this algorithm are limited as well. Other directions of Active Learning are information seeking or curious agents, which look very promising in the near future. If we think in how a person interacts with the world when pursuing a task, the number of possibilities to choose is almost infinite. Choosing the correct task to receive a positive feedback can not be done by randomly sampling uncertain decisions, as most of the possible decisions will not produce any kind of feedback for that task. We think that agents that learn to be curious in a less "bounded" environment is currently a very promising research line.

One-shot learning is a very hard problem and we still do not understand the mechanisms that humans use to recognize new objects so easily. By now, one-shot learning is similar to a domain adaptation setting, where the training classes are the source domain, and the test classes are the target domain. It would be interesting to exploit new domain adaptation methods in order to better generalize to new one-shot samples.

Bibliography

[1] Aurélien Bellet, Amaury Habrard, and Marc Sebban. "A survey on metric learning for feature vectors and structured data". In: *arXiv preprint arXiv:1306.6709* (2013).

[2] Michael M Bronstein et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[3] Corinna Cortes and Vladimir Vapnik. "Support vector machine". In: *Machine learning* 20.3 (1995), pp. 273–297.

[4] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[5] David K Duvenaud et al. "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems*. 2015, pp. 2224–2232.

[6] Harrison Edwards and Amos Storkey. "Towards a neural statistician". In: *arXiv preprint arXiv:1606.02185* (2016).

[7] Meng Fang, Yuan Li, and Trevor Cohn. "Learning how to Active Learn: A Deep Reinforcement Learning Approach". In: *arXiv preprint arXiv:1708.02383* (2017).

[8] Li Fei-Fei, Rob Fergus, and Pietro Perona. "One-shot learning of object categories". In: *IEEE transactions on pattern analysis and machine intelligence* 28.4 (2006), pp. 594–611.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *arXiv preprint arXiv:1703.03400* (2017).

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[11] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[12] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[13] Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. "Multi-class active learning for image classification". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 2372–2379.

[14] Ajay J Joshi, Fatih Porikli, and Nikolaos P Papanikolopoulos. "Scalable active learning for multiclass image classification". In: *IEEE transactions on pattern analysis and machine intelligence* 34.11 (2012), pp. 2259–2273.

[15] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition". In: *ICML Deep Learning Workshop*. Vol. 2. 2015.

[16] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. "Learning Active Learning from Real and Synthetic Data". In: *CoRR* abs/1703.03365 (2017). URL: http://arxiv.org/abs/1703.03365.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[18] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[19] Brenden M Lake, Ruslan R Salakhutdinov, and Josh Tenenbaum. "One-shot learning by inverting a compositional causal process". In: *Advances in neural information processing systems*. 2013, pp. 2526–2534.

[20] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[21] Yann LeCun et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. 1990, pp. 396–404.

[22] David D Lewis and William A Gale. "A sequential algorithm for training text classifiers". In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc. 1994, pp. 3–12.

[23] Xin Li and Yuhong Guo. "Adaptive active learning for image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 859–866.

[24] Yujia Li et al. "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[25] Akshay Mehrotra and Ambedkar Dukkipati. "Generative Adversarial Residual Pairwise Networks for One Shot Learning". In: *arXiv preprint arXiv:1703.08033* (2017).

[26] Marvin Minsky and Seymour Papert. "Perceptrons." In: (1969).

[27] Nikhil Mishra et al. "Meta-Learning with Temporal Convolutions". In: *arXiv preprint arXiv:1707.03141* (2017).

[28] Tsendsuren Munkhdalai and Hong Yu. "Meta Networks". In: *arXiv preprint arXiv:1703.00837* (2017).

[29] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[30] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning". In: (2016).

[31] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[32] Nicholas Roy and Andrew McCallum. "Toward optimal active learning through monte carlo estimation of error reduction". In: *ICML, Williamstown* (2001), pp. 441–448.

[33] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[34] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[35] Franco Scarselli et al. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80.

[36] Ozan Sener and Silvio Savarese. "A Geometric Approach to Active Learning for Convolutional Neural Networks". In: *arXiv preprint arXiv:1708.00489* (2017).

[37] Burr Settles. "Active learning literature survey". In: *University of Wisconsin, Madison* 52.55-66 (2010), p. 11.

[38] Burr Settles and Mark Craven. "An analysis of active learning strategies for sequence labeling tasks". In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2008, pp. 1070–1079.

[39] Claude E Shannon. "A note on the concept of entropy". In: *Bell System Tech. J* 27.3 (1948), pp. 379–423.

[40] Jake Snell, Kevin Swersky, and Richard S Zemel. "Prototypical Networks for Few-shot Learning". In: *arXiv preprint arXiv:1703.05175* (2017).

[41] Panagiotis Stanitsas et al. "Active convolutional neural networks for cancerous tissue recognition". In: ICIP. 2017.

[42] Sainbayar Sukhbaatar, Rob Fergus, et al. "Learning multiagent communication with backpropagation". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2244–2252.

[43] Simon Tong and Daphne Koller. "Support vector machine active learning with applications to text classification". In: *Journal of machine learning research* 2.Nov (2001), pp. 45–66.

[44] Oriol Vinyals et al. "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3630–3638.

[45] Keze Wang et al. "Cost-effective active learning for deep image classification". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2016).

[46] Min Wang et al. "Active learning through density clustering". In: *Expert Systems with Applications* 85 (2017), pp. 305–317.