Image Feature Learning with Genetic Programming

Stefano Ruberto[1], Valerio Terragni[2], and Jason H. Moore[1]

[1] Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, USA
`stefano.ruberto@pennmedicine.upenn.edu` `jhmoore@upenn.edu`
[2] Faculty of Informatics, Università della Svizzera Italiana USI, Lugano, Switzerland
`valerio.terragni@usi.ch`

Abstract. Learning features from raw data is an important topic in machine learning. This paper presents Genetic Program Feature Learner (GPFL), a novel generative GP feature learner for 2D images. GPFL executes multiple GP runs, each run generates a model that focuses on a particular high-level feature of the training images. Then, it combines the models generated by each run into a function that reconstructs the observed images. As a sanity check, we evaluated GPFL on the popular MNIST dataset of handwritten digits, and compared it with the convolutional neural network LeNet5. Our evaluation results show that when considering smaller training sets, GPFL achieves comparable/slightly-better classification accuracy than LeNet5. However, GPFL drastically outperforms LeNet5 when considering noisy images as test sets.

Keywords: Genetic Programming · Semantic GP · Feature learning, · Image classification

1 Introduction

Feature learning [49] is an important topic in machine learning, as it powers many classification and knowledge discovery techniques. Such techniques need numeric representations of raw data (i.e., features) that are computationally convenient to process [49]. Feature learning becomes a key task when dealing with raw high-dimensional data (e.g., 2D images, videos and sounds) [49], which lack well-defined features [16, 23, 26]. Manually identifying features from high-dimensional data is often infeasible because it requires expensive human-labor and domain knowledge [1, 36]. As such, automatic feature learning techniques have gained much attention [30, 49].

Most recent automatic feature learners are implemented as (multi-layer) neural networks [49]. However, in principle, an automatic feature learner based on GP would entail two important advantages: (i) GP often does not need large training sets to learn competitive models [33]; and (ii) GP is generally robust to noisy data [27]. Recently, we have seen the first GP feature learners for 2D images [9, 25, 38]. These approaches emulate the behavior of a neural-network *autoencoder* [18], with multiple GP-evolved models that reconstruct (encode and decode) the pixels of an image [38]. Evolving a dedicated model to reconstruct a single pixel has two main issues: (i) it is computationally expensive, especially with high-resolution images; and (ii) it ignores the (important) spatial relations of pixels, as it evolves each model independently from the ones of adjacent pixels.

This paper presents a **GP F**eature **L**earner, called **GPFL**[3], to learn high-level features from 2D images. There are two main differences between GPFL and the previous GP feature learners [9, 25, 38]. First, GPFL does not evolve as many models as the pixels in the image, but one model for each high-level feature of the image. Second, GPFL learns high-level features leveraging the spatial relations of pixels, as it uses the pixel coordinates as inputs of the models.

In a nutshell, GPFL takes in input a training dataset of images and outputs a model represented as a function f_{gp} that, given a 2D coordinate (c_1, c_2), returns a pixel value p (i.e., $f_{gp}(c_1, c_2) = p$). As such, GPFL is a generative and unsupervised feature learner. Under the hood, GPFL follows the *dynamic target* approach SGP-DT [41,42] that executes multiple GP runs (called external iterations). Each external iteration evolves a population of models driven by a dynamic "*target*" that changes at each iteration. Each target is defined as the *residual errors* of the reconstructed images between the previous and current iterations. As such, the next iteration will focus on the characteristics of the images that the previous iteration did not approximated well. Each external iteration outputs a model (called partial model) that focuses on a specific high-level feature of the images. When all external iterations terminate, GPFL creates the final model f_{gp} by combining with *linear scaling* [17,43] all the partial models.

As a sanity check, we evaluated GPFL on the popular MNIST dataset of 2D images representing handwritten digits [6]. We evaluated how well the models trained by GPFL capture the relevant high-level features of the MNIST images. Towards this goal, we implemented a classifier that uses the reconstruction error of f_{gp} to classify the MNIST digits. We compared the GPFL-based classifier with LeNet5 [19, 21], the well-known DNN specific for MNIST.

When trained with smaller training sets and evaluated with all 10,000 images in the MNIST test set, GPFL-based classifier achieves a median classification accuracy that is comparable or better than LeNet5. For example, when trained with a dataset composed of ten images for each digit, the GPFL-based classifier has a median classification accuracy of 82.81%, while LeNet5 of 80.67%.

We also evaluated the noise robustness of GPFL by corrupting the 10,000 images of the test set with five noise levels (of salt type). GPFL always outperforms LeNet5 for all five levels (with a classification accuracy improvement up to +40.85%). These are important results, considering that GPFL is one of the first genetic programming attempts to learn high-level features from 2D images.

The remainder of this paper is structured as follows. Section 2 discusses the related work. Section 3 describes the GPFL approach. Section 4 presents our experiments. Section 5 concludes the paper.

[3] We presented a preliminary version of this work in a poster paper [40].

2 Background and Related Work

GPFL aims to automatically learn high-level features from 2D images. Following previous work, we use low-level features to refer to the pixel values, and high-level features to refer to conglomerations of related low-level features.

Deep Neural Networks (DNNs) [20] are often used to learn high-level features from high-dimensional data with great success [2,6]. Because developing DNNs requires labor-intensive architecture engineering, researchers have investigated GP approaches (e.g., NEAT techniques [3–5,8,10,11,34,35]) to automate the architecture engineering activity of DNNs. These techniques show promising but still limited results, as finding an optimized DNN architecture largely remains a human activity. Instead of leveraging GP to explore the space of possible DNN architectures, GPFL is a GP feature learner detached from DNNs.

Most GP feature learners for 2D images discover high-level features using hand-crafted or domain-specific features as building blocks [1,16,23,26,36]. For example, Speeded Up Robust Features (SURF) [7], Histogram of Oriented Gradients (HoG) [31], Gist features [32] and Scale-Invariant Feature Transform (SIFT) [50]. Differently, GPFL learns high-level features from low-level ones (i.e., pixel values) without requiring human-crafted or domain-specific features.

At the best of our knowledge, there are only three attempts of GP feature learners for 2D images that discover high-level features directly from low-level ones [25,29,38]. Such attempts, following the success of NN-based autoencoders [18,28,45], use GP to emulate the classical autoencoder architecture with *encoder→code→decoder*. The *encoder* component learns a compact representation (called code) of the low-level features in input. The *decoder* component uses the learned high-level features (i.e., the code) to reconstruct an approximation of the input. We now discuss these three attempts.

Rodriguez-Coayahuitl et al. proposed *Structured Layered GP* (SLGP) [37,38], which evolves two distinct populations. One population encodes the pixels in input and outputs the *code* (i.e, latent space). The other population decodes the *code* into the reconstructed image. SLGP generates as many encoding GP trees as the size of *code* and as many decoding GP trees as the number of pixels.

McDermott proposed an autoencoder GP similar to SLGP [29]. Differently from SLGP, it relies on two *multi-value linear GP* components [9], one for the *encoder* and one for the *decoder*.

Lensen et al. proposed GPMaL [24], GP technique for *manifold learning* [46], which relates to both SLGP and McDermott's approach. Manifold learning aims to reduce the dimensions of raw data. This is similar, in principle, to the *encoder* component of most autoencoders, which transforms the input into a lower dimensional code (latent space). GPMaL resembles the *encoder* of SLGP, as it also uses as many GP trees as the number of dimensions of the latent space (the *code* size in SLGP). A later version of GPMaL [25] relies on a Pareto front technique to dynamically select the number of dimensions of the latent space.

Similarly to GPFL, these three techniques are generative approaches that reconstruct 2D images. However, GPFL differs substantially. First, they simulate the classical NN autoencoder architecture with two distinct components: *encoder* and *decoder*. Conversely, GPFL does not follow the NN autoencoder architecture, and thus it avoids altogether the issue of aligning the two components. Second, previous GP feature learners evolve a GP model for each pixel, which is computationally expensive when dealing with high-resolution images, and does not directly consider the spatial relations among pixels. Notably, GPFL is the first GP feature learner that directly relies on spatial information (pixel coordinates). Third, they require that the number of high-level features (i.e., *code* size) is chosen in advance. Instead, one can run GPFL with an arbitrary number of external iterations (i.e., number of high-level features) and stop at the desired reconstructed error. Moreover, one can re-run GPFL to obtain additional high-level features without discarding the previously learned features.

Although the three previous attempts have been evaluated with MNIST, GPFL is the only one known to classify all ten MNIST digits.

3 Genetic Programming Feature Learning (GPFL)

Most high-dimensional data found in nature exhibit correlations among low-level features expressed by the extra dimensions. For 2D images, such correlations are the spatial relations among pixels (being the space the extra dimension). The spatial position of pixels can be extremely useful to express relevant high-level features, as it characterizes the intrinsic properties of the image itself: Two images with pixels of identical values but of different spatial positions can represent two radically different concepts. Indeed, humans recognize patterns and objects by relying heavily on spatial relations [22].

This paper presents GPFL, a GP feature learner for 2D images that relies on both the pixels values and their spatial positions. GPFL outputs a function f_{gp} (a GP-evolved model) that, given a 2D coordinate (c_1, c_2), returns a pixel value p (i.e., $f_{gp}(c_1, c_2) = p$). As such, given all coordinates, f_{gp} reconstructs an image.

Each model (GP individual) is a mathematical (tree-like) expression, with

(i) *non-terminal symbols*: algebraic operations ($+, -, \cdot$, the protected division, *Min* and *Max*) and trigonometric operations (*sine* and *cosine*)
(ii) *terminal symbols*: variables (the coordinates c_1 and c_2) and decimal constants (*ERC* between -1 and 1).

This dictionary of symbols allows GPFL to evolve continuous functions with the coordinates c_1 and c_2 as independent variables. As such, the produced models can encode spatial relations among pixels. This is because the continuity property entails relations on adjacent low-level features (i.e., pixels). Because the protected division, Min and Max symbols introduce discontinuity, the models can also encode spatial relations that are difficult to model in a single continuous function. For instance, by combining multiple (continuous) functions.

The key challenge of using spatial information for feature learning is their variability among images that represent the same concept. For example, when classifying handwritten digits, “1” or “I” are two popular styles for writing the number one. These styles have different, albeit similar, spatial relations. A

Algorithm 1: GPFL implements the *dynamic-target framework* (SGP-DT [41]), ** marks the lines representing the novel aspects of GPFL

```
input   : $\hat{y}[c_1][c_2] \in \hat{\mathbb{Y}}$: training 2D images (H×W matrices of pixels)
          number of external ($N_{ext}$) and internal ($N_{int}$) iterations
output  : $f_{gp}$ : final regression model
1     Function GPFL
2**     $target \leftarrow \hat{\mathbb{Y}}$
3       $partialModels \leftarrow [\cdots]$
4       for $ext\text{-}iter$ from 1 to $N_{ext}$ do
5         $\mathcal{P} \leftarrow$ GET-RANDOM-INITIAL-POPULATION
6         for $int\text{-}iter$ from 1 to $N_{int}$ do
7           for each $\mathcal{I} \in \mathcal{P}$ do
8**           $\mathcal{I}_{ls} \leftarrow$COMPUTE-FITNESS-AND-LS$(target, \mathcal{I})$        // see Algorithm 2
9           $\mathcal{P}' \leftarrow \emptyset$
10          add ELITE($\mathcal{P}$) to $\mathcal{P}'$
11          while $\mathcal{P}'$ is not full do
12            $\mathcal{I}_{ls} \leftarrow$ TOURNAMENT-SELECTION($\mathcal{P}$)
13            add MUTATE($\mathcal{I}_{ls}$) to $\mathcal{P}'$
14          $\mathcal{P} \leftarrow \mathcal{P}'$
15        $\mathcal{I}^{\star}_{ls} \leftarrow$ GET-BEST-INDIVIDUAL($\mathcal{P}$)        // partial model
16        add $\mathcal{I}^{\star}_{ls}$ to $partialModels$
          /* the new target is computed as the residual errors of each image */
17**      for each $i$ from 1 to $size(target)$ do
18**        for each $c_1$ from 1 to $H$ do
19**          for each $c_2$ from 1 to $W$ do
20**            $target[i][c_1][c_2] \leftarrow \mathcal{I}^{\star}_{ls}(c_1, c_2) - target[i][c_1][c_2]$
21**    return $f_{gp} \leftarrow \sum_{\mathcal{I}^{\star}_{ls} \in partialModels} \mathcal{I}^{\star}_{ls}$        // linear combination of partial models
```

single non-parametric function cannot output different pixel values for the same coordinate, and thus cannot encode both styles. GPFL addresses the challenge by parameterizing f_{gp}, so that changing the parameter values reproduces the observed variability. GPFL defines such parameters as the coefficients of a linear combination of multiple GP-evolved models (called partial models), each focusing on a specific high-level feature of the images.

Algorithm 1 describes GPFL's approach. It has three inputs: (i) the training images ($\hat{\mathbb{Y}}$); (ii) the number of external iterations (N_{ext}) (i.e, the number of partial models); and (iii) the number of internal iterations (N_{int}) (i.e, the number of generations that GPFL uses to optimize each partial model). GPFL relies on *linear scaling* [17] to construct f_{gp} as a linear combination of the partial models.

To generate the partial models, GPFL implements the *dynamic-target* approach SGP-DT [41] that evolves multiple models driven by a *target* that changes at each external iteration. SGP-DT initializes the *target* with the training set (line 2, Algorithm 1). At each external iteration (lines 4–20), SGP-DT evolves a population of models ($\mathcal{P}$) to identify one (partial model $\mathcal{I}^{\star}_{ls}$ line 15) that best approximates the current *target*. SGP-DT evolves $\mathcal{P}$ using a variant of the classical GP algorithm (lines 6–13) that does not use any form of crossover [41]. Ruberto et al. have shown that such a variant is effective with the dynamic-target approach [41]. At each new external iteration, SGP-DT computes the new target as the residuals errors of the current target and the best model $\mathcal{I}^{\star}_{ls}$ (lines 17–20).

Algorithm 2: GPFL's fitness function

```
input  : $target$: set of 2D images and $\mathcal{I}$: individual
output : $\mathcal{I}_{ls}$ encoded with fitness score and linear scaling coefficients
22 Function COMPUTE-FITNESS-AND-LS
23     $scores \leftarrow [\cdots]$                                  // vector of score for each image in target
24     for each $\hat{y}$ in $target$ do
25         $y_p \leftarrow [\cdots][\cdots]$                          // predicted image
26         for each $c_1$ from 1 to $H$ do
27             for each $c_2$ from 1 to $W$ do
28                 $y_p[c_1][c_2] \leftarrow \mathcal{I}(c_1, c_2)$
29         $\langle a, b\rangle \leftarrow$ COMPUTE-AND-STORE-LS-COEFFICIENTS$(y_p, \hat{y})$
30         $\mathcal{I}_{ls} \leftarrow a + b \cdot \mathcal{I}$                     // linear scaling
31         $scores[\hat{y}] \leftarrow$ COMPUTE-MEAN-SQUARED-ERROR$(\mathcal{I}_{ls}, \hat{y})$
32     $fitness\text{-}score(\mathcal{I}_{ls}) \leftarrow \frac{\sum scores[i]}{size(scores)}$   // arithmetic mean of the scores
33     return $\mathcal{I}_{ls}$
```

Ruberto et al. defined the dynamic-target framework SGP-DT for the numerical symbolic regression domain [41]. We now describe how GPFL adapts it for learning high-level features from 2D images. We mark with ** the lines of Algorithm 1 that correspond to the novel aspects of GPFL. First and foremost, GPFL proposes a novel fitness function, which is specific to our problem at hand (Function COMPUTE-FITNESS-AND-LS, lines 22-33, Algorithm 2). Second, GPFL generates the new target by computing the error residuals by differencing images (lines 17-20, Algorithm 1). Third, GPFL constructs the final model using a linear combination. Differently, SGP-DT uses a validation set, which does not apply in our case. We now describe in details these three novel aspects of GPFL.

Fitness function. GPFL invokes Function COMPUTE-FITNESS-AND-LS (Algorithm 2) for each individual in the current population (line 8, Algorithm 1). The function takes in input (i) the current *target*, which are the 2D residual images at the current iteration; and (ii) the individual $\mathcal{I}$. Line 33 of Algorithm 2 returns $\mathcal{I}_{\mathrm{ls}}$, the individual $\mathcal{I}$ with its fitness score and linear scaling coefficients (a and b). Note that a and b are different for each image in *target*. Intuitively, the fitness score captures how well an individual approximates the current target.

The function starts by initializing at empty the vector of scores (line 23 of Algorithm 2), which will populate with the prediction errors of $\mathcal{I}_{ls}$, for each image $\hat{y}$ in *target*. Given an image $\hat{y}$, GPFL generates the predicted image y_p by computing the function $f_{gp}(c_1, c_2)$ prescribed by the individual $\mathcal{I}$ for all the $H \times W$ coordinates c_1 and c_2 in the image $\hat{y}$. GPFL assigns the results of $f_{gp}(c_1, c_2)$ (predicted value) to $y_p[c_1, c_2]$ (line 28, Algorithm 2). Given y_p and $\hat{y}$, GPFL computes the coefficient a and b with the linear scaling technique [17] (line 29, Algorithm 2). Following Keijzer [17], we compute the linear scaling of an individual $\mathcal{I}$ as follows (where $\overline{\hat{y}}$ is the arithmetic mean of $\hat{y}$)[4]

$$\mathcal{I}_{ls} = a + b \cdot \mathcal{I} \tag{1}$$

$$\text{where} \quad a = \overline{\hat{y}} - b \cdot \overline{y_p} \quad \text{and} \quad b = \frac{\sum_{i=1}^{n}[(\hat{y}_i - \overline{\hat{y}}) \cdot (y_{pi} - \overline{y_p})]}{\sum_{i=1}^{n}[(y_{pi} - \overline{y_p})^2]} \tag{2}$$

[4] The cost of computing the linear scaling coefficients is $\mathcal{O}(\mid \hat{\mathbb{Y}} \mid \cdot \mid \mathcal{P} \mid)$.

Following classical GP approaches, we rely on the Mean Squared Error (MSE) between y_p and $\hat{y}$ to compute the scores (line 31, Algorithm 2). Because y_p and $\hat{y}$ are images, MSE measures the average squared difference between the predicted value $y_p[c_1, c_2]$ and the actual value $\hat{y}[c_1, c_2]$, for each coordinate (c_1, c_2). Being a quadratic function, MSE gives more weight to the pixels with a greater difference. As such, during the first external iterations, GPFL focuses on those elements of the images that lead to greater errors.

After the function analyzes all residual images in *target*, it computes the fitness score of $\mathcal{I}_{ls}$ as the arithmetic mean of the scores (line 32, Algorithm 2). The rationale of choosing the arithmetic mean is to consider equally important all the images in *target.* The fitness score is a number from zero to infinite. Because it represents an error, the lower the score the fitter the individual.

Constructing the new target. To construct the new target, GPFL scans all the pixel coordinates and computes the difference between the current pixel value and the one predicted by the best model $\mathcal{I}^{\star}_{ls}$ (lines 17–20, Algorithm 1). As such, the next iteration will focus on the characteristics of the images that the previous iteration did not approximate well. Note that the linear scaling coefficients are different for every image and were previously computed by the Function COMPUTE-FITNESS-AND-LS.

Constructing the final model. GPFL constructs the final model with a linear combination of all the partial models (line 21, Algorithm 1). Intuitively, by combining all partial models we are summing all the estimates of the residuals, and thus obtaining a function f_{gp} that well approximates the training images in input. Notably, f_{gp} is a parametric function with a and b as parameters.

4 Experiments

This section describes a series of experiments to evaluate how well the models trained by GPFL capture the most relevant high-level features of 2D images. Because given enough external iterations GPFL can achieve an arbitrary lower reconstruction error, we opted to evaluate GPFL with classification accuracy instead. In fact, linear scaling guarantees that the reconstruction error (i.e., RMSE) monotonically decreases [17]. This happens because GPFL re-computes the linear scaling coefficients when reconstructing each test image.

We built a naïve classifier that relies on GPFL for classifying MNIST digits (Algorithm 3), and compared with the DNN LeNet5 [19, 21][5]. We experimented with smaller MNIST training sets and with noisy MNIST test sets to evaluate the generalizabilty and robustness of the models, respectively. This is a common experimental setup [13] for the *few-shot learning problem* [12].

Dataset. The MNIST database of handwritten digits [48] comprises a training set of 60,000 examples, and a test set of 10,000 examples. Each example is a grayscale numerical bitmap image of 28x28 pixels representing a handwritten

[5] When comparing the classification accuracy of GPFL and LeNet5, we computed the p-values with the non-parametric pairwise *Wilcoxon rank-sum test* [15].

Algorithm 3: GPFL-based MNIST naïve classifier

```
input  : $\hat{\mathbb{Y}}$ MNIST training set, $S$ ensembles size
output : $ensembles$ for each digit
1  Function TRAINER
2      for each digit from 0 to 9 do
3          for each $i$ from 1 to $S$ do
4              $ensembles[digit][i] \leftarrow$ GPFL ($\hat{\mathbb{Y}}[digit]$, $N_{ext}$=100, $N_{int}$=50)
5      return $ensembles$

input  : $ensembles$ for each digit returned by the trainer, $\hat{y}$ image to classify
output : predicted digit of $\hat{y}$
6  Function PREDICTOR
7      for each digit from 0 to 9 do
8          $\hat{y}_{rc} \leftarrow [\cdots][\cdots]$                // reconstructed image initialiated at empty
9          for each $i$ from 1 to $S$ do
10             $ensembles[digit][i]_{ls} \leftarrow$ COMPUTE-LS($ensembles[digit][i]$, $\hat{y}$)
11             $\hat{y}_{rc} \leftarrow \hat{y}_{rc} + reconstruction(ensembles[digit][i]_{ls})$
12         $average\text{-}image \leftarrow \hat{y}_{rc}/N$                // average of each pixel
13         $error[digit] \leftarrow$ MSE($\hat{y}$, $average\text{-}image$)        // mean square error
14     return $digit \leftarrow \underset{digit \in \{0 \cdots 9\}}{\text{argmin}} \{error[digit]\}$
```

digit from 0 to 9. MNIST is widely-used as a standard benchmark in the ML community [6, 47]. Even now, MNIST is often the first dataset that researchers use to validate their algorithms [2–5, 8, 11]. From the MNIST training set of 60,000 images, we constructed three variants of smaller size, with five (MNIST-5), ten (MNIST-10), and one hundred (MNIST-100) images for each digit. Because there are ten digits (0 to 9), the three variants contain 50, 100 and 1,000 examples, respectively. We constructed such variants by randomly sampling the MNIST training set. To avoid selection biases, we repeated the sampling process 30 times for each of the three variants, obtaining 90 datasets in total. We stored them on disk to train GPFL and LeNet5 with exactly the same datasets.

A GPFL-based classifier. To evaluate how well GPFL learns relevant high-level features that characterize the images in input, we constructed a *naïve classifier* (Algorithm 3) that classifies unseen MNSIT digits relying on the models that GPFL produces. The classifier splits the training images $\hat{\mathbb{Y}}$ into ten partitions ($\hat{\mathbb{Y}}[digit]$) according to their digit label. Then, it uses GPFL to train multiple models (called ensemble) for each of the partitions. We use the *ensemble method* to mitigate the stochasticity of GP, which can lead to models of arbitrary performance. Note that, although GPFL is unsupervised, the classifier is supervised because it splits the training set according to the digit labels.

The naïve classifier has two components: the TRAINER and the PREDICTOR (see Algorithm 3). The predictor takes in input the test image to classify and the list of ensembles precomputed by the trainer. To predict the digit of the test image, the predictor reconstructs the image multiple times, one for each ensemble. Internally, the predictor obtains each reconstructed image by averaging the pixels outputted by the models. Then, it returns the digit corresponding to the ensemble that yielded the lowest reconstruction error. When reconstructing the test image $\hat{y}$, we recompute the linear scaling parameters that best approximate $\hat{y}$.

Table 1: Classification accuracy of GPFL and LeNet5 on 10,000 test images

# external iter. (N_{ext})	ensembles size (S)	GPFL median accuracy %			learning rate	LeNet5 median accuracy %		
		MNIST-5	MNIST-10	MNIST-100		MNIST-5	MNIST-10	MNIST-100
20	50	73.78%	81.21%	90.16%	0.1	**74.48%**	**80.67%**	92.67%
30	50	74.18%	82.53%	91.26%	0.01	73.88%	80.58%	**92.76%**
40	50	**74.49%**	**82.81%**	91.66%	0.001	73.52%	79.92%	92.28%
60	50	73.82%	82.70%	**91.84%**	0.0001	73.90%	79.48%	91.49%
100	50	73.27%	82.19%	91.26%				
40	1	68.36%	76.32%	85.07%				
40	30	74.33%	82.63%	91.59%				

We decided to rely on a naïve classifier (as opposed to more sophisticated approaches, e.g., SVM and Random forest) to isolate our main contribution. So that the effectiveness of the classification (Algorithm 3) can be mainly attributed to the quality of the high-level features that GPFL identified.

GPFL configuration. Following the dynamic target framework SGP-DT [41], we configured GPFL as follows. Fifty internal iterations (N_{int}). One hundred the population size. The *ramped-half-and-half* initialization generates trees with a depth that ranges from 1 to 4 (line 5, Algorithm 1). The probability of mutation is 100%, and the maximum depth of the subtrees generated by the mutation operators is five. The probability of a sub-tree mutation happening at the leaf level is 70%. We set no limits on the number of nodes and on the depth of the trees. We set size of the tournament selection to two, and the elitism size to one.

We ran GPFL with different combinations of N_{ext} (number of external iterations) and S (the ensembles size) and choose the best ones. Table 1 gives the median accuracy of GPFL on the 30 datasets of each variant, using the original test set of 10,000 images. The different combinations of N_{ext} and S give similar accuracy results, except for the combination $N_{ext} = 40$ and $S = 1$, which has the lowest performance. This confirms the importance of an ensembles approach. Table 1 marks in bold the highest median accuracy for each of the datasets. In our experiments, we will use the corresponding values of N_{ext} and S.

LeNet5 configuration. We compared GPFL with the Convolutional Neural Network (CNN) LeNet5 [19, 21], the most used baseline for MNIST [13, 14]. We implemented LeNet5 with the TensorFlow framework. To be sure that our implementation was correct, we confirmed that when trained with all the 60,000 training images and validated with the 10,000 test images, our implementation achieves the advertised classification accuracy of 99% [19, 21]. We released our datasets, models and implementation and we welcome external validation [39].

A key hyper-parameter of CNNs is the *learning rate* that controls how much the weights are adjusted with respect to the loss gradient [20]. The lower the value, the slower the training. The original LeNet5 uses 0.1 as learning rate, which might be inadequate in our case since we have smaller training sets. Table 1 shows the median accuracy of LeNet5 on our datasets with different learning rates (0.1, 0.01, 0.001, and 0.0001), using the original test set of 10,000 images. Table 1 marks in bold the highest median accuracy for each of the three datasets.

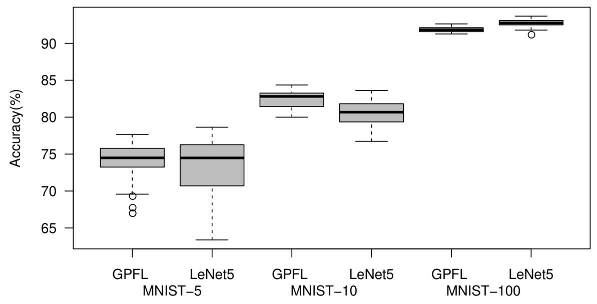

Fig. 1: GPFL and LeNet5 classification accuracy (best configurations)

Classification accuracy on 10,000 test images. Figure 1 shows the box-plot of the classification accuracy distributions of GPFL and LeNet5 with their best configurations (see Table 1). On MNIST-5, GPFL and LeNet5 achieve a similar median accuracy of ∼74.50%, but GPFL exhibits less variance. This is the only non-statistically significant result (p-value 0.597). On MNIST-10, GPFL outperforms LeNet5 (p-value $5.14{\cdot}10^{-6}$) with a median accuracy of 82.81% and 80.67%, respectively. On MNIST-100, GPFL underperforms LeNet5 (p-value $1.17{\cdot}10^{-5}$) with a median accuracy of 91.84% and 92.76%, respectively.

As expected, with the increasing of the training size, both techniques attain better classification accuracy. Moreover, because smaller datasets might lack representative training cases, the variance increases when the training size decreases. Despite that the architecture of LeNet5 was specifically designed for the MNIST dataset [19,21], GPFL's results are comparable with LeNet5 on MNIST-5, and better than LeNet5 on MNIST-10. This demonstrates that, given small training sets, GPFL learns the relevant high-level features of MNIST images. This is an important result considering that GPFL's architecture is agnostic to MNIST.

Classification accuracy on 10,000 noisy test images. We added random noise to the MNIST test set of 10,000 images to compare the noise resilience of GPFL and LeNet5. We considered five levels of salt noise L%: 5%, 10%, 20%, 30%, 40%. L specifies the percentage of randomly selected pixels in each image whose values turn into 255 (white color). We decided to use salt noise because (in our case) is more disruptive than the salt-and-pepper noise. In fact, the majority of pixels in MNIST images are black (background color). The left matrix in Figure 3 exemplifies the five noise levels (Columns 5%, 10%, 20%, 30%, 40%).

Table 2 shows the median accuracy for each combination of training sets (MNIST-5, MNIST-10 and MNIST-100) and noise levels (5%, 10%, 20%, 30%, 40%). GPFL always outperforms LeNet5 for every combination of training sets and noise levels. The comparison is always statistical significant (p-values $< 10^{-6}$), except when comparing GPFL and LeNet5 trained on MNIST-5 and tested with noise level 5% (p-value 0.121). For noise level 5%, the difference between the median accuracy of GPFL and LeNet5 ranges from +1.11% to +2.84%. With the increasing of the noise level, the difference constantly grows. For noise level 40%, the difference ranges from +30.51% to +40.85%.

Table 2: Median accuracy with 10,000 noisy test images

salt noise level %	MNIST-5		MNIST-10		MNIST-100	
	GPFL	LeNet5	GPFL	LeNet5	GPFL	LeNet5
5	73.54%	72.43%	82.01%	79.18%	91.01%	89.75%
10	72.79%	68.95%	80.99%	75.56%	89.55%	80.49%
20	69.47%	56.78%	78.02%	63.45%	84.98%	57.37%
30	65.31%	41.10%	73.00%	49.26%	76.42%	37.72%
40	59.18%	28.67%	65.73%	35.99%	65.49%	24.64%

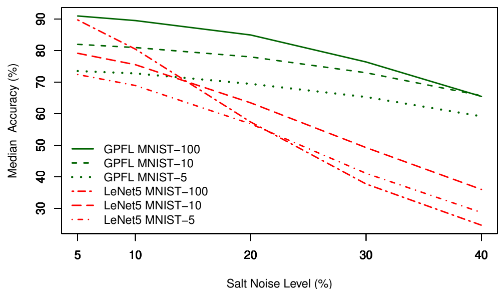

Fig. 2: Median classification accuracy degradation at the increasing of noise level.

Figure 2 plots the median accuracy trend at the noise level increases. For all the three MNIST variants, LeNet5 accuracy degrades much faster than the one of GPFL. Interestingly, GPFL always outperforms LeNet5.

For the lowest noise level (5%), LeNet5 trained on MNIST-100 (denoted by LeNet5$_{100}$) outperforms LeNet5$_5$ and LeNet5$_{10}$. This is an expected result, because (in principle) the larger the training set the highest the classification accuracy. However, the performance of LeNet5$_{100}$ drastically decreases when the noise level increases. For noise level 30% and 40%, LeNet5$_{100}$ performs significantly worse than LeNet5$_5$ and LeNet5$_{10}$. This result can be due to LeNet5$_{100}$ has *"overfitted the clean data"*: MNIST-100 has the largest size and it requires more epochs to converge. As such, when the noise level increases, the difference between the training and the test sets also increases. Intuitively, the higher this difference, the lower the classification accuracy. Tsipras et al. had similar conclusions when testing recent DNNs with noisy MNIST test sets [44].

The classification accuracy of GPFL$_{100}$ degrades at the increasing of the noise, but at a much slower pace. Only at noise level 40%, GPFL$_{100}$ achieves a similar classification accuracy with GPFL$_{10}$. Analogously to LeNet5, GPFL$_{100}$ has *"overfitted the clean data"*: MNIST-100 has the largest size and the highest number of external iterations ($N_{\text{ext}} = 60$ vs 40 see Table 1).

Reconstruction results. The right matrix in Figure 3 shows ten images from the MNIST test set and their GPFL's reconstructions at various numbers of external iterations (i.e., partial models). These are the results of GPFL trained on MNIST-100 with ensembles size $S = 50$. Column N_{ext} shows the images that GPFL reconstructs using the linear combination of the first N_{ext} partial models, that is $f_{gp}(\mathit{original}) = \sum_{i=1}^{N_{\text{ext}}} \mathit{partialModels}[i]$. With a low value of N_{ext}, the reconstructed images focus on the macro characteristics of the images. For

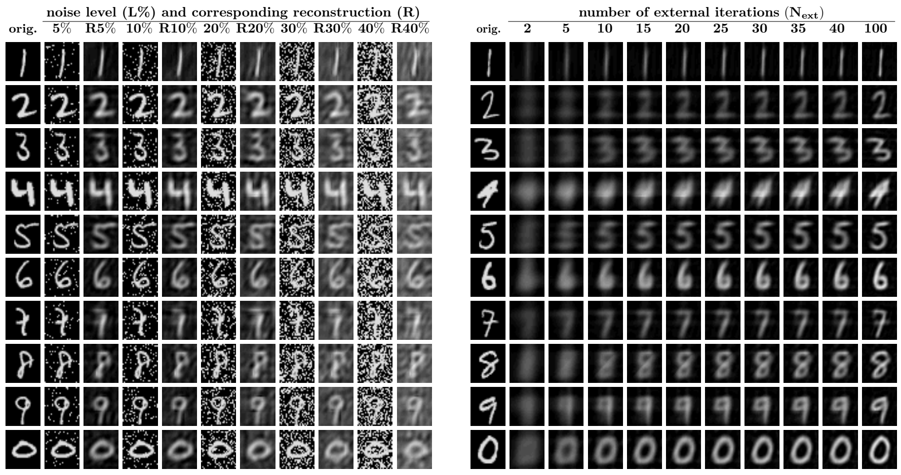
noise level (L%) and corresponding reconstruction (R)
orig. 5% R5% 10% R10% 20% R20% 30% R30% 40% R40%
number of external iterations (N_ext)
orig. 2 5 10 15 20 25 30 35 40 100

Fig. 3: Examples of reconstructed images of GPFL (with noise on the left).

instance, the images of Column $N_{\text{ext}} = 2$ show clouds of dust that resemble the shape of the digits. When N_{ext} increases, the finer details gradually appear because GPFL focuses on the residual errors of previous iterations. As Figure 3 exemplifies, the process looks like a progressive cleansing of the images.

The left matrix of Figure 3 shows ten images of the MNIST test set, their noisy versions ($L\%$) and their reconstructions (R) using GPFL$_{100}$. The reconstructed digits are recognizable even at noise level 40%. However, the reconstructions of the digits two and zero show some artifacts originated by an uneven distribution of the noise that GPFL interpreted as high-level features.

Size of the final solutions. The average size of f_{gp} with $N_{int} = 50$ and $N_{ext} = 40$ is 4,587 ($\pm$4.6%), which is the number of nodes in the tree-like representation of f_{gp}. Recall that GPFL constructs f_{gp} by assembling the partial models with a linear combination. As such, after 50 internal iterations the resulting partial models have an average size of 115 nodes (i.e., $4,587/40 = 115$).

5 Conclusion

This paper presented GPFL, a GP technique to learn high-level features from 2D images. Differently from previous GP feature learner attempts, GPFL does not simulate the behavior of Deep Neural Networks (DNNs) whatsoever. Our novel GP approach can handle more complex classification tasks than previous attempts. Our experiments with MNIST show that GPFL has a competitive edge with LeNet5 when considering small training sets and noisy test sets.

Note that, we are not claiming that GPFL *is a valid alternative to DNNs for learning high-level features from 2D images.* In fact, MNIST is the (simple) de-facto standard benchmark for a first sanity check only. Moreover, we compared GPFL with LeNet5 that (although being specific to MNIST) is not the most recent DNN-based feature learner. However, GPFL demonstrates that a GP feature learner can lead to interesting results that are worth investigating further.

References

1. Albukhanajer, W.A., Briffa, J.A., Jin, Y.: Evolutionary multiobjective image feature extraction in the presence of noise. IEEE Transactions on Cybernetics **45**(9), 1757–1768 (2015). https://doi.org/10.1109/TCYB.2014.2360074
2. Alvear-Sandoval, R.F., Sancho-Gómez, J.L., Figueiras-Vidal, A.R.: On improving cnns performance: The case of MNIST. Information Fusion **52**, 106–109 (2019)
3. Baldominos, A., Saez, Y., Isasi, P.: Evolutionary convolutional neural networks: An application to handwriting recognition. Neurocomputing **283**, 38–52 (2018). https://doi.org/10.1016/j.neucom.2017.12.049
4. Baldominos, A., Saez, Y., Isasi, P.: Model selection in committees of evolved convolutional neural networks using genetic algorithms. In: Intelligent Data Engineering and Automated Learning. pp. 364–373. IDEAL '18 (2018). https://doi.org/10.1007/978-3-030-03493-1_39
5. Baldominos, A., Saez, Y., Isasi, P.: Hybridizing evolutionary computation and deep neural networks: An approach to handwriting recognition using committees and transfer learning. Complexity (2019). https://doi.org/10.1155/2019/2952304
6. Baldominos, A., Saez, Y., Isasi, P.: A survey of handwritten character recognition with MNIST and EMNIST. Applied Sciences **9**(15), 3169 (2019)
7. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: Proceedings of the European Conference on Computer Vision. pp. 404–417. ECCV '06 (2006)
8. Bochinski, E., Senst, T., Sikora, T.: Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In: Proceedings International Conference on Image Processing. pp. 3924–3928. ICIP '17 (2017). https://doi.org/10.1109/ICIP.2017.8297018
9. Brameier, M.F., Banzhaf, W.: Linear genetic programming. Springer (2007). https://doi.org/10.1007/978-0-387-31030-5_1
10. Butterworth, J., Savani, R., Tuyls, K.: Evolving indoor navigational strategies using gated recurrent units in NEAT. In: Proceedings of the Companion of Genetic and Evolutionary Computation Conference. pp. 111–112. GECCO '19 (2019). https://doi.org/10.1145/3319619.3321995
11. Davison, J.: DEvol: Automated deep neural network design via genetic programming. (2020), https://github.com/joeddav/devol
12. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(4), 594–611 (2006)
13. George, D., Lehrach, W., Kansky, K., Lázaro-Gredilla, M., Laan, C., Marthi, B., Lou, X., Meng, Z., Liu, Y., Wang, H., et al.: A generative vision model that trains with high data efficiency and breaks text-based captchas. Science **358**(6368), 2612 (2017)
14. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. pp. 1135–1143 (2015)
15. Haynes, W.: Wilcoxon rank sum test. Encyclopedia of systems biology pp. 2354–2355 (2013)
16. Impedovo, S., Mangini, F.: A novel technique for handwritten digit classification using genetic clustering. In: Proceedings of International Conference on Frontiers in Handwriting Recognition. pp. 236–240. ICFHR '12 (2012). https://doi.org/10.1109/ICFHR.2012.167

17. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Proceedings of the European Conference on Genetic Programming. pp. 70–82. EuroGP '03 (2003). https://doi.org/10.1007/3-540-36599-0_7
18. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE journal **37**(2), 233–243 (1991)
19. LeCun, Y.: Lenet-5, convolutional neural networks (2020), http://yann.lecun.com/exdb/lenet
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. vol. 86, pp. 2278–2324. Ieee (1998)
22. Legge, G.E., Foley, J.M.: Contrast masking in human vision. Josa **70**(12), 1458–1471 (1980)
23. Lensen, A., Al-Sahaf, H., Zhang, M., Xue, B.: Genetic programming for region detection, feature extraction, feature construction and classification in image data. In: Proceedings of the European Conference on Genetic Programming. pp. 51–67. EuroGP '16 (2016)
24. Lensen, A., Xue, B., Zhang, M.: Can genetic programming do manifold learning too? In: Proc. of the European Conference on Genetic Programming. pp. 114–130. EuroGP '19 (2019). https://doi.org/10.1007/978-3-030-16670-0_8
25. Lensen, A., Zhang, M., Xue, B.: Multi-objective genetic programming for manifold learning: Balancing quality and dimensionality. Genetic Programming and Evolvable Machines (2020). https://doi.org/10.1007/s10710-020-09375-4
26. Liu, L., Shao, L., Li, X.: Evolutionary compact embedding for large-scale image classification. Information Sciences **316**, 567–581 (2015). https://doi.org/10.1016/j.ins.2014.06.030
27. López, U., Trujillo, L., Martinez, Y., Legrand, P., Naredo, E., Silva, S.: RANSAC-GP: Dealing with outliers in symbolic regression with genetic programming. In: Proceedings of the European Conference on Genetic Programming. pp. 114–130. EuroGP '17 (2017)
28. Makhzani, A., Frey, B.J.: Winner-Take-All Autoencoders. In: Advances in Neural Information Processing Systems. pp. 2791–2799 (2015)
29. McDermott, J.: Why is auto-encoding difficult for genetic programming? In: Proceedings of the European Conference on Genetic Programming. pp. 131–145. EuroGP '19 (2019). https://doi.org/10.1007/978-3-030-16670-0_9
30. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.: Reading digits in natural images with unsupervised feature learning. Google technical report (2011)
31. Neumann, L., Matas, J.: Real-time scene text localization and recognition. In: Proceedings of Conference on Computer Vision and Pattern Recognition. pp. 3538–3545. CVPR '12 (2012)
32. Oliva, A., Torralba, A.: Building the gist of a scene: The role of global image features in recognition. Progress in Brain Research pp. 23–36 (2006)
33. Orzechowski, P., La Cava, W., Moore, J.H.: Where are we now?: A large benchmark study of recent symbolic regression methods. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1183–1190. GECCO '18 (2018). https://doi.org/10.1145/3205455.3205539
34. Papavasileiou, E., Jansen, B.: An investigation of topological choices in FS-NEAT and FD-NEAT on xor-based problems of increased complexity. In: Proceedings of the Companion of Genetic and Evolutionary Computation Conference. pp. 1431–1434. GECCO '17 (2017). https://doi.org/10.1145/3067695.3082497

35. Peng, Y., Chen, G., Singh, H., Zhang, M.: NEAT for large-scale reinforcement learning through evolutionary feature learning and policy gradient search. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 490–497. GECCO '18 (2018). https://doi.org/10.1145/3205455.3205536
36. Perez, C.B., Olague, G.: Genetic programming as strategy for learning image descriptor operators. Intelligent Data Analysis **17**(4), 561–583 (2013). https://doi.org/10.3233/IDA-130594
37. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J.: Structurally layered representation learning: Towards deep learning through genetic programming. In: Proceedings of the European Conference on Genetic Programming. pp. 271–288. EuroGP '18 (2018)
38. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J.: Evolving autoencoding structures through genetic programming. Genetic Programming and Evolvable Machines **20**(3), 413–440 (2019)
39. Ruberto, S., Terragni, V., Moore, J.H.: GPFL replication package. experimental data of GPFL and source code of Lenet5. (Apr 2020). https://doi.org/10.5281/zenodo.3899891
40. Ruberto, S., Terragni, V., Moore, J.H.: Image feature learning with a genetic programming autoencoder. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '20 (2020)
41. Ruberto, S., Terragni, V., Moore, J.H.: SGP-DT: Semantic genetic programming based on dynamic targets. In: Proceedings of the European Conference on Genetic Programming. pp. 167–183. EuroGP '20 (2020)
42. Ruberto, S., Terragni, V., Moore, J.H.: SGP-DT: Towards effective symbolic regression with a semantic GP approach based on dynamic targets. In: Proceedings of the Genetic and Evolutionary Computation Conference (Hot Off the Press track), GECCO '20 (2020)
43. Ruberto, S., Vanneschi, L., Castelli, M.: Genetic programming with semantic equivalence classes. Swarm and Evolutionary Computation **44**, 453 – 469 (2019). https://doi.org/10.1016/j.swevo.2018.06.001
44. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy (2018)
45. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the International Conference on Machine Learning. pp. 1096–1103. ICML '08 (2008)
46. Wang, J., Zhang, Z., Zha, H.: Adaptive manifold learning. In: Advances in Neural Information Processing Systems. NIPS '05
47. Yadav, C., Bottou, L.: Cold case: The lost MNIST digits. In: Advances in Neural Information Processing Systems. pp. 13443–13452. NIPS '19 (2019)
48. Yann LeCun, C.C., Burges, C.: Mnist handwritten digit database (2020)
49. Zheng, A., Casari, A.: Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists. O'Reilly Media, Inc. (2018)
50. Zhou, H., Yuan, Y., Shi, C.: Object tracking using sift features and mean shift. Computer Vision and Image Understanding **113**(3), 345–352 (2009)