Experiments with SVM and CNN

1st Cleiton Moya de Almeida
PESC/COPPE
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
cma@cos.ufrj.br

***Abstract*—This work explores Support Vector Machines and Convolutional Neural Networks. In the first experiment, we apply SVM with different kernels to classify the Keel Bananas Dataset. The performance is compared with a Decision Tree Model. We achieve best results with the RBF kernel (90%) and worst with linear kernel (55%), whereas the Decision Tree Model exhibits accuracy in order of 78%. In the second experiment, CNN models are constructed to to classify the MNIST dataset. We propose seven different models, and the best one achieves accuracy of 99%. Our conclusion is that both SVM and CNN are powerful machine learning methods.**

***Index Terms*—machine learning, suppport vector machine, deep learning, convolutional neural network**

I. Introduction

This document describes the final work developed in the discipline CPS849 - Computational Intelligence II offered during the first quarter of 2020 by professors C. E. Pedreira and C. G. Marcelino.

In the first experiment, we explore Support Vector Machines (SVM) models to classify the Keel Bananas Dataset. Results are compared with a Decision Tree Model.

In the second experiment, Convolutional Neural Networks (CNN) models are constructed to classify the MNIST dataset.

A. Computational Framework

The experiments was executed with the following computational framework:

- Language: Python 3.6;
- IDE: Spyder 4.1.5;
- Main libraries: scikit-learn 0.23.2, Keras 2.3.1, Pandas 1.1.1, Matplotlib 3.3.1, Seaborn 0.10.1;
- CPU: Intel Core i7-6700 3.40Ghz;
- GPU: NVIDIA GeForce GTX 745.

The code used in both experiments are attached to this work.

II. Background

A. Support Vector Machine

The main idea of SVM classifier is to find an hyperplane that maximizes the margin of data separation.

The problem of margin optimization can be formulated in the following primal form [1]: Giving training vector $\mathbf{x} \in \Re^d$, and a vector of labels $\mathbf{y}$, with $y_n \in \{-1, 1+\}$

CNPq

$$\begin{aligned} \underset{\mathbf{w}, b, \xi}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^\intercal \mathbf{w} + C\sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & y_n(\mathbf{w}^\intercal \phi(x_n) + b) \geq 1 - \xi_i n = 1, \dots, N, \\ & \xi_n \geq 0 \qquad n = 1, \dots, N \end{aligned} \tag{1}$$

with $\mathbf{w} \in \Re$, $b \in \Re$ and $\boldsymbol{\xi} \in \Re$.

The dual problem (Lagrange formulation) is:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\intercal Q \boldsymbol{\alpha} - (\mathbf{1})^\intercal \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^\intercal \boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_n \leq C \quad n = 1, \dots, N \end{aligned} \tag{2}$$

where C is the penalization term regarding margin error and Q is the positive semi-definite matrix in the form $Q_{nm} \equiv y_n y_m K(x_n, x_m)$. $K(x_n, x_m) = \phi(x_n)^\intercal \phi(x_m)$ is the kernel function.

In this work we use the following kernel functions:

- Linear: $\langle x, x' \rangle$;
- Polynomial: $(\gamma\langle x, x' \rangle + r)^d$;
- RBF: $\exp(-\gamma \|x - x'\|)^2$;
- Sigmoid: $\tanh(-\gamma\langle x, x' \rangle + r)$.

Once the optimization problem is solved, $\boldsymbol{w}$ is computed using the Lagrangian $\boldsymbol{\alpha}$:

$$\boldsymbol{w} = \sum_{x_n \text{is SV}} \alpha_n y_n \mathbf{x}_n \tag{3}$$

The support vectors are all x_n such that the lagrangian a_n is not null.

A great advantage of SVM is that, knowing the number of support vectors, we can bound the out-of-sample error having only in-sample information:

$$\mathbb{E}[E_{out}] \leq \frac{\mathbb{E}[\text{\# of SV}]}{N-1}. \tag{4}$$

B. Decision Trees

Decisions Trees (DT) are supervised learning algorithms that learn logical expressions (which can be represented as a set of "IF THEN" rules) from the features values of the instances [3].

The main algorithms used for decision tree learning are the ID3 and its successor C4.5. CART (Classification and

Regression Trees), which is similar to C4.5, is used by the sikit-learn library [4].

The basic idea of the ID3 algorithm (and its variants) is to infer a decision tree searching (greedily) attributes that most contribute to a a information gain, e.g. cross entropy. The tree are constructed from top (root) to down (leaves). Each node contains a inferred rule [4].

One of the main advantages of Decision Trees is their easy interpretation, once the learned tree can be visualized. In the other hand, they tends to easily over-fit if no regularizer is used [3] [4].

C. Convolutional Neural Network

Convolutional Neural Networks (CNN) was first developed by Yann LeCon and others in 1998 inspired on researches on the visual cortex of animals. The key concept is the *receptive field*, which describes the link between parts of the visual field and individual neurons that process the information [5]. Image recognition is the typical application, but CNNs can also be used in time series problems.

In convolutional neural networks we have the following key layers and concepts, ilustrated in Fig. 1) [5]:

- *Convolutional layers*: these layers take an image and passes (“convolution”) a “logistic regression” over the whole image. The features are combined and the original image are transformed in another one, with reduced dimension, called feature map.
- *Local receptive fields* (kernels): the “convolution” are done by receptive fields, generating the feature maps. It is common to use more than one receptive field per layer. Thus, one image is transformed to several feature maps. Receptive fields often use an activation function called *rectified linear unit*, or ReLU. The ReLU of x is simple the maximum value of 0 and x;
- *Feature maps*: images resulted after the convolution.
- *Max-pooling layer*: this layer takes a pool size as hyper-parameter, usually 2x2. Then, the input image is divided in small areas (e.g. 2x2) and the pixel with the highest value is taken. Thus, a new image with the half of size of the original one is produced. The idea of the max-pooling is “important information in a picture is seldom contained in adjacent pixels (this accounts for the ‘pick-one-out-of-four’ part), and it is often contained in darker pixels (this accounts for using the max)” [5].
- *Flattening layer*: convert images to vectors.
- *Fully connected (dense) layer*: like a feed-forward layer, uses all the outputs of previous layer and generates new ones.

Comparing to the traditional multi-layer percepetron (MLP), CNN can be viewed as a regularized version of the MLP. In the MLP, usually each neuron is fully connected to all other neurons from the next layer. Thus, MLP are more prone to overfitting data [6].

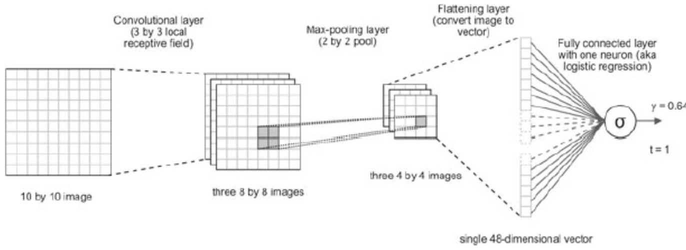
Convolutional layer (3 by 3 local receptive field)
Max-pooling layer (2 by 2 pool)
Flattening layer (convert image to vector)
Fully connected layer with one neuron (aka logistic regression)
y = 0.6453
σ
t = 1
10 by 10 image
three 8 by 8 images
three 4 by 4 images
single 48-dimensional vector

Fig. 1. Typical architecture of a CNN. Source: [5]

D. Cross validation for model selection

In machine learning models we have two types of parameters: parameters that are automatically learned by model during training and parameters we have to manually set before the training. We refer to last one as “hyperparameters” of the model. In the SVM model, for example, the penallity constant C is an hyperparameter.

Unfortunately, usually there is no straightforward methods to tuning the hyperparameters, and in practical we have to optimize them by try-and-error. So, we have to use a validation set to evaluate different models (based on different hyperparameters) and choose the best one. The problem is, the more we use the same validation set to choice among different models, more contaminated this sets becomes, leading to a unrealistic estimate of out-of-sample error.

One way to minimizes this problem,specially if we do not have too much available data, is to use cross-validation. In particular, a popular method is the K-fold cross validation.

The idea of k-fold is to part the dataset $\mathcal{D}$ (with N instances) into K sets of size N/K. Then, for each iteration i (i from 1 to K), we use $\mathcal{D}_i$ as validation dataset and the others $K-1$ partitions for training [2].

Using cross-validation, we can select the hyperparameters in the following way [2]:

1) For each hyperparameter combination (model), we perform a K-fold validation and compute the mean error of the model E_{cv} considering the K folds;
2) We select the model $m*$ with the minimum E_{cv} and use this $E_{cv}(m*)$ as an estimate of the out-of-sample error (E_{out});
3) We use the model $m*$ and all the data $\mathcal{D}$ to obtain the final hypothesis g_{m*}.

III. Experiment #1

In this experiment we explore the capacity of SVM model to classify the synthetic database “Banana” using different kernels: linear, polynomial, sigmoid and RBF. Also, we compare the performance of SVM with a Decision Tree model.

A. The Dataset

The “Banana” synthetic data set from Keel[1] contains 5,300 instances grouped in several clusters with a banana shape.

[1]https://sci2s.ugr.es/keel/dataset.php?cod=182#sub

Instances are labeled in two classes (-1, +1) and have two features. Fig. 2 shows a scatter plot of the dataset.

One characteristic of Banana dataset is that are data already normalized with mean 1 an standard deviation 0 (Table I). Also, data are well balanced, with 55% of instances in class -1 and 45% in class +1.

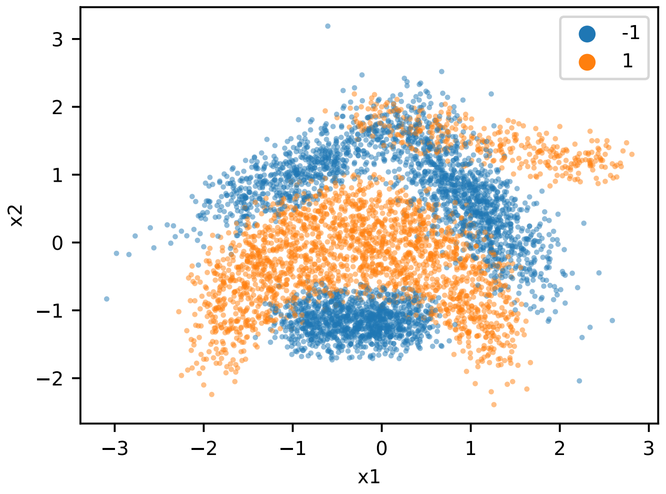

Fig. 2. Bananas dataset - scatter plot

TABLE I
BANANA DATASET FEATURES

	x1	x2
mean	0.000016	0.000018
std	0.999880	1.000038
min	-3.090000	-2.390000
max	2.810000	3.190000

B. Experiment Design

In this experiment he have tested four different kernels. For each kernel, in order to tune the hyper-parameters, we perform a cross-validated grid search using k-fold, as described in section II-D, with $k = 2, 5, 10$.

The Table II shows the hyper-parameters of each SVM kernel as well as the range of these hyper-parameters we consider in the grid search. For the polynomial kernel, the hyper-parameter *coef0* corresponds to the parameter r in the kernel equation from section II-A, and *degree* corresponds to the parameter r. For the sigmoid kernel, *coef0* corresponds to the r parameter.

For the hyperparameter grid definition, first we had tried to use the same range ($[10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3]$) for C and γ in all models. However, during experiments we have noted the polynomial kernel took too time or even did not converge using $C > 1$ and $\gamma > 1$.So, we limited the range.

C. Results

The Table III shows the results from the experiment. On this table, we use the metric mean *accuracy*, which corresponds to the fraction of correct classified instances in the validation set. This metric is computed considering the average of the accuracy for each fold executed, so its directly related to the out-sample-error E_{out} and E_cv described on section II-D. The table also shows the best hyperparameters estimated for the model.

TABLE II
SVM - HYPERPARAMETERS GRID SEARCH

Kernel	C	gamma	coef0	degree
Linear	(-2, 2, 5)[a]	(-2, 1, 4)	-	-
Poly	(-2, 0, 3)	(-2, 0, 3)	[0, 1, 5]	[2, 3, 5]
RBF	(-3, 3, 7)	(-3, 3, 7)	-	-
Sigmoid	(-3, 3, 7)	(-3, 3, 7)	[0.01, 0.5, 1]	-

[a]logspace(-2, 2, 5) = [0.01, 0.1, 1, 10, 100]

The model that best performed was SVM with RBF kernel (model #9), achieving accuracy in the order of 90%. The worst performance, about 55%, was get with the linear kernel (model #3).

In order to better analyze these performances, we plotted the support vectors and the decision function limits of these two models in figures 3 and 4. Following the recommendation discussed on section II-D, we generate the final hypothesis g_{m*} based on the entire data set $\mathcal{D}$.

TABLE III
EXP. #1 - EVALUATED MODELS

#	Model	K-fold	Accuracy	Best Hyper.[a]
1	SVM Linear	2	0.552 ± 0.005	$[0.01, 0.01, -, -]$
2	SVM Linear	5	0.552 ± 0.009	$[0.01, 0.01, -, -]$
3	SVM Linear	10	0.552 ± 0.030	$[0.01, 0.01, -, -]$
4	SVM Poly	2	0.903 ± 0.002	$[0.01, 1, 5, 3]$
5	SVM Poly	5	0.903 ± 0.006	$[0.01, 1, 5, 3]$
6	SVM Poly	10	0.905 ± 0.007	$[0.1, 1, 5, 3]$
7	SVM RBF	2	0.906 ± 0.002	$[1, 1, -, -]$
8	SVM RBF	5	0.906 ± 0.009	$[1, 1, -, -]$
9	SVM RBF	10	0.907 ± 0.008	[1, 1, -, -]
10	SVM Sigm	2	0.558 ± 0.027	$[1, 10, 1, -]$
11	SVM Sigm	5	0.553 ± 0.009	$[1000, 100, 0.01, -]$
12	SVM Sigm.	10	0.554 ± 0.031	$[1000, 100, 0.01, -]$
12	D. Tree	10	0.779 ± 0.022	table IV

[a]For SVM, hyperparameters: [C, gamma, coef0, degree]

In the Fig. 3, we can see that SVM with RBF kernel correctly delimit the “bananas” clusters. Support vectors clearly delimit with high accuracy these clusters. In the other hand, Fig. 4 shows that SVM with a linear kernel leads to a high number of support vectors, and then to high order E_{out} (as expected through 4).

The reason to SVM with linear kernel being not capable to classify the dataset is due to non-linear characteristic of the dataset. Clearly, data is not linearly separable in $\Re^2$, so we must take the data to a higher dimension space in order to get them linearly separable. This is only achieved using a non-linear kernel function, such as RBF.

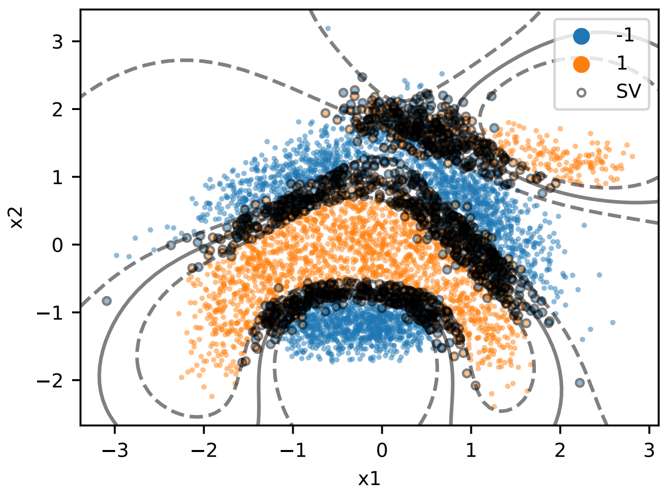

Fig. 3. Model #9 (RBF) - Decision limits and support vectors

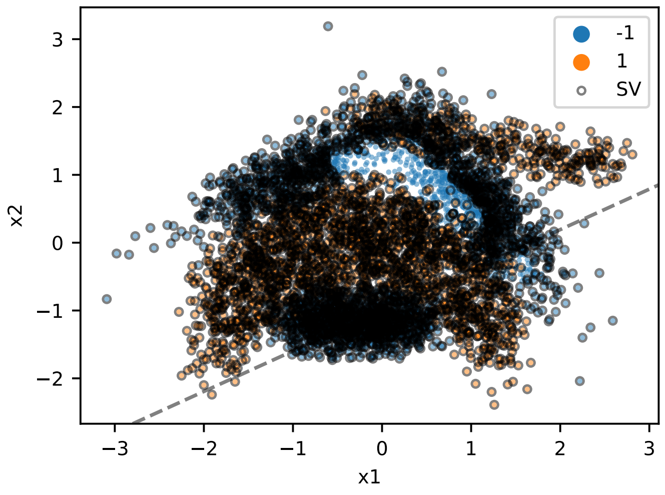

Fig. 4. Model #3 (Linear) - Decision limits and support vectors

D. Comparing to Decision Tree

In order to compare the performance of SVM with another machine learning model, we choose the Decision Tree Model (explained in section II-B).

It is worth to mention that our first idea was to compare the performance of SVM RBF with a RBF Network, given the similarity of these two models. However, unfortunately, at this time neither scikit-learn or Keras provide built-in models for RBF Networks. Beyond this fact, we choose Decision Tree because their simplicity to understood the algorithm, as well as the results, as discussed in section II-B. Also, decision trees are used as base for popular ensemble methods like Random Forest.

We apply the same cross-validation and hyper-parameter tuning strategy used previously. The grid search is shown on IV. With DT, we achieved accuracy in the order of 77%. Given the simplicity of the method, we consider this a good result.

In order to visualize the learned rules, we plotted the decision surface (5) and tree (6). As we can see, the learned rule is very simple.

Comparing to SVM with RBF kernel, DT is not capable to produces complex decision limits. Maybe better results can be achieved if others hyperparameters or strategies, but this is not our focus in this work.

TABLE IV
DECISION TREE - 10-FOLD CROSS-VALIDATION

Hyperparameter	Range	Best
criterion	['gini','entropy']	'gini' -
max_depht	$[3, 4, 5]$	3
max_leaf_nodes	$[3, 5]$	5
min_impurity_decrease	$[0.01, 0.05, 0.1]$	0.01 -
min_sample_leaf	$[1, 2]$	1

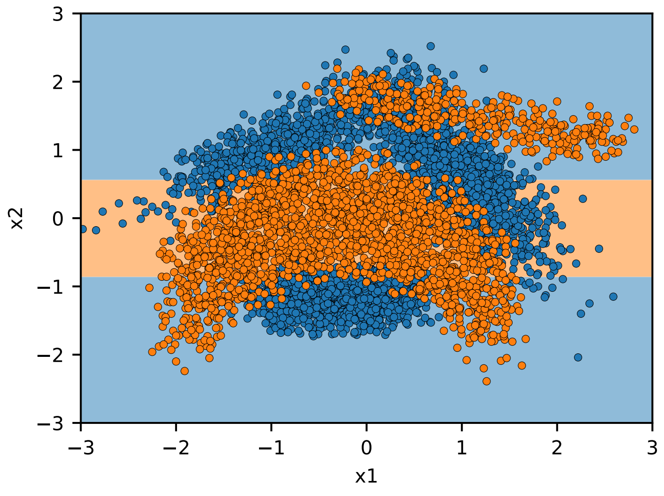

Fig. 5. DT Model - Decision Surface

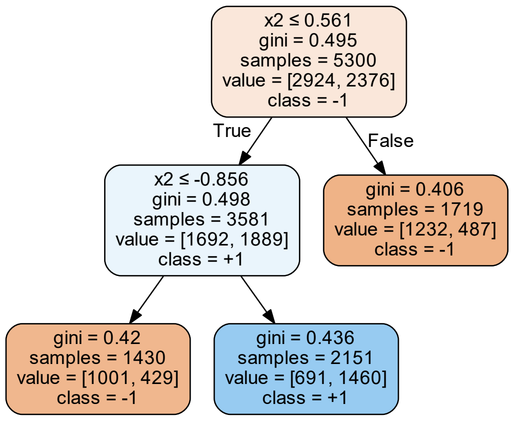

Fig. 6. DT Model - Learned Tree

IV. Experiment #2

In the second experiment, we explore CNN model to classify the MNIST dataset. To implement the CNN we use the Keras API[2].

A. The Dateset

The MNIST database[3] has 70,000 examples of handwritten digits, with 60,000 instances for training and 10,000 instances for validation. The dataset was constructed from the original NIST database, but the digits have been size-normalized and centered in a fixed-size image. Its a popular database and it is built-in available in the `keras.dataset` module.

Each image is represented by a $28x28$ matrix, where each entry contains the pixel value from 0 (white background) to 255 (black background). Fig. 7 shows some instances of the dataset.

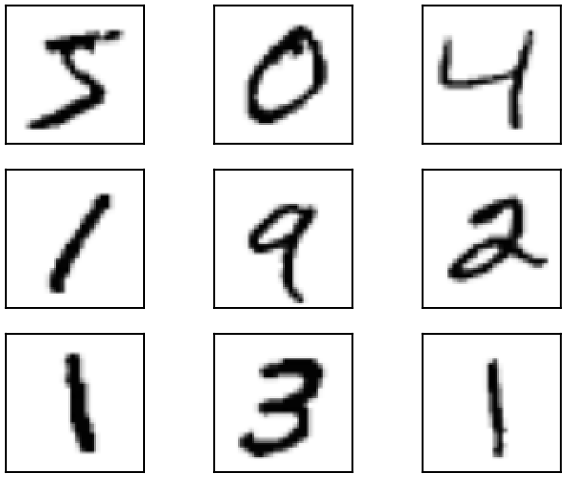

Fig. 7. Some Instances of the MNIST Dataset

B. Data Pre-Processing

The Keras API works with images in the size $M \times N \times 1$, so in the first step we perform a reshape of the original dataset. Furthermore, we normalize the data in order to get pixels value in the range $[0, 1]$, instead of $[0, 255]$.

C. Models Design

For the network design, we considered two main architectures:

1) Conv. 2D $\rightarrow$ Maxpooling $\rightarrow$ Dropout $\rightarrow$ Flatten $\rightarrow$ Dense
2) (Conv. 2D $\rightarrow$ Maxpooling)^2 $\rightarrow$ Dropout $\rightarrow$ Flatten $\rightarrow$ Dense

The functionality of these layers are discussed on section II-C. In addition, we used Dropout that is not a real layer: it is a modification in the connections of two layers, including a dropout rate. This dropout rate act as a regularizer.

[2]https://keras.io/

[3]http://yann.lecun.com/exdb/mnist/

Based on these two main architectures, we constructed seven models (Table V) changing the following variables:

- Number of feature maps ("functions") on each convolutional layer: 32 or 64;
- Loss function: Mean Squared Error (MSE) or Categorical Cross Entropy (CTE);
- Optimizer algorithm: Stochastic Gradient Descendant (SGD) or Adam.

TABLE V
Exp. #2 - Evaluated Models

#	Conv2D[a]	Loss	Optimizer
1	32	MSE[b]	SGD[c]
2	32	CTE[d]	SGD
3	32	CTE	Adam
4	32 + 32	CTE	Adam
5	32 + 64	CTE	Adam
6	64 + 32	CTE	Adam
7	64 + 64	CTE	Adam

[a]Num. of functions in Conv2D layers
[b]Mean Squared Error
[c]Stochastic Gradient Descendant
[d]Categorical Cross-Entropy

In all the seven models, we considered the following equal hyper-parameters:

- Kernel size (# of receptive fields): $3x3$;
- Activation function: ReLU;
- Pool size (Maxpooling layer): $2x2$;
- Dropout: 0.25;

Beyond the dropout, we included in the models two additional regularization:

- Early Stopping, with parameter "patience" $= 1$; and
- L2 weight regularization in the Conv2D layers of size 64: $\lambda = 5 \times 10^4$.

D. Experiment Design

Once we propose to evaluate only 7 models, without grid search, and given that the size of MNIST dataset is relative big (we have 10,000 instances for validation), we did not perform cross-validation in this experiment. The training was done with the original MNIST training set and the validation with the original validation set.

In all runs, we used 5 epochs of training and a bath size of 16.

In order to get fast training, we trained the models using a GPU (section I-A).

E. Results

The model with best accuracy was model 6#, and the worst result was presented by model #1. The accuracy of all models are shown in Table VI.

Fig. 8 shows the evolution of loss and accuracy, regarding the number of epochs, for Model #6.

From results, we can note that using a cost function of type Categorical Cross Entropy, instead of Mean Squared Error,

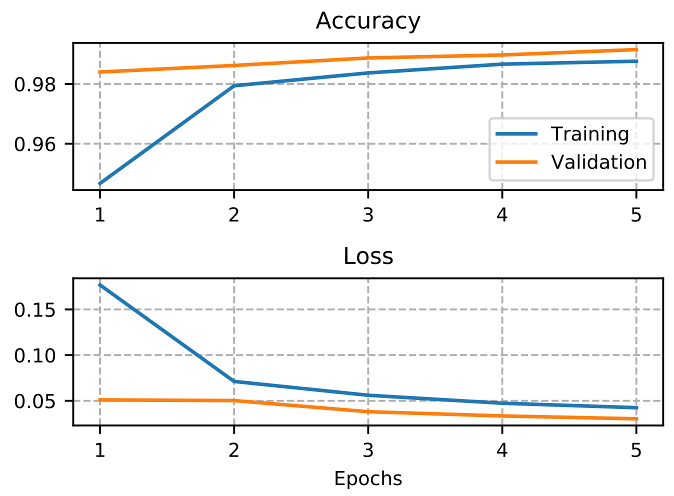

Fig. 8. CNN Model #6 - Accuracy and Loss Evolution

TABLE VI
EXP. #2 - RESULTS

Model #	Accuracy	Loss	Time (min.)
1	0.9037	0.1546	1.4
2	0.9666	0.1147	1.7
3	0.9832	0.0498	1.8
4	0.9906	0.0270	2.9
5	0.9901	0.0452	2.7
6	**0.9915**	**0.0303**	2.8
7	0.9895	0.0517	3.0

leaded to a significant increasing in the accuracy. Also, Adam optimizer better performed in this task.

Regarding the number of layers, we can note that the models with two convolutional layers (models #4, 5, 6 and 7) performed better than the models with only one. However, the number of feature maps did not leads to majors improvements.

For all the 7 models, with a bath size of 16, 5 epochs was sufficient to achieve good accuracy, as shown in Fig. 8. However, during the experiments we noted that, increasing bath size to 64, for example, the number of required epochs increases.

V. CONCLUSIONS

This work explored two of the most used and powerful machine learning algorithms: Support Vector Machines and Convolutional Neural Networks.

In the first experiment, we explored Support Vector Machines with different kernels to classify the Keel Banana Dataset. The best result was achieved with RBF kernel (accuracy of order 90%). Using linear kernel, in the other hand, lead us to the worst accuracy (55%).

Yet regarding experiment #1, we compared the results of SVM with a Decision Tree. This model presented an accuracy of 78%, inferior to SVM with polynomial and RBF kernels. So, we conclude that SVM is a powerful method for classification, even for complex data sets.

In the second experiment, we explored Convolutional Neural Networks models to classify images of the MNIST dataset. The constructed models performed very well. The best achieved accuracy was superior to 99%. So we concluded that CNNs are very well suitable for image pattern recognition.

ACKNOWLEDGMENT

The author would like to thanks CNPq for the financial support received.

REFERENCES

[1] Y. Abu-Mostafa. Learning From Data Lecture 15: Kernel Methods, Lectures Notes, available at https://work.caltech.edu/lectures.html.
[2] Y. Abu-Mostafa, M. Magdon-Ismail, H. Lin. Learning from Data: a Short Course. Section 4.3: Validation. AMLbook.com, 2012. pp. 137-153.
[3] T. M. Mitchell, Machine Learning. New York: McGraw-Hill Science, 1997, pp.52-77.
[4] Scikit-Learning Developers. Decision Trees. Retrieved September 17, 2020, from https://scikit-learn.org/stable/modules/tree.html.
[5] S. Skansi. Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence. Chapter 6: Convolutional Neural Networks. Springer, 2018. pp. 121-133.
[6] Wikipedia Contributors. Convolutional neural network. In Wikipedia, The Free Encyclopedia. Retrieved September 18, 2020, from https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=976842800