Implementation of a CNN for Asterism Classification in Carte du Ciel Astrographic Maps

Lasko M. Laskov(✉) and Radoslav Radev

Informatics Department, New Bulgarian University, Sofia, Bulgaria
llaskov@nbu.bg

Abstract. Carte du Ciel together with Astrographic Catalogue form a 19th century huge international astronomical project whose goal was to map the stars in the visible sky as faint as 14th magnitude. The result, in the form of astrographic plates and their paper copies – astrographic maps, are stored and investigated in many astronomy institutes worldwide.

The goal of our study is to develop image processing and pattern recognition techniques for automatic extraction of astronomical data from the digitized copies of the astrographic maps. In this paper we present the design and implementation of a convolutional neural network (CNN) for automatic classification of stars images in scanned Carte du Ciel astrographic maps. We do not use any deep learning frameworks to build our model, and we focus on the low-level implementation of the CNN. Also, we provide comparison of our implementation with an implementation based on PyTorch.

Keywords: Deep learning · Convolutional neural network · Astrographic maps

1 Introduction

Carte du Ciel was the major part, along with *Astrographic Catalogue*, of a vast international project that started in 1887 and continued until 1962 in many observatories around the world [5]. The purpose of the project was to create a catalogue of the positions of the stars in the visible sky that are as faint as 11th magnitude. Another goal was to map the relative positions of stars of 14th magnitude and brighter [14].

The data collected was in the form of glass photographic plates taken by the telescopes, called *astrographic plates* [2]. The dimensions of the photographed field was $2^\circ \times 2^\circ$, and their location in the sky were selected in such way, so that the corner of each plate lies at the center of its neighbor [5]. The physical dimensions of the plates were $12\,\mathrm{cm}^2$, and each plate contains triple exposure with an approximate duration 20 min each. The reason for the triple exposure was to

T. Zlateva and G. Tuparov (Eds.): CSECS 2023, LNICST 514, pp. 102–114, 2023.
https://doi.org/10.1007/978-3-031-44668-9_8

be able to distinguish the images of the stars from the images of other types of celestial bodies and different type of noise that can result in the photographic plates (see Fig. 1b).

The *astrographic maps* themself are paper copies that were produced from the astrographic plates using photogravure on copper plates. They represent negative image in which the background in white, and light objects, such as stars, are in black. Each star is represented by a triple image that is called *asterism* by Fresneau [2] (see Fig. 1a).

(a) (b)

Fig. 1. (a) A fragment of an astrographic map containing six asterisms, and (b) an asterism degraded by noise.

Originally, asterisms were used to calculate the exact coordinates of the stars in the astrographic maps. However, they can also contain important information about the star that is represented. A significant difference in the Gaussian distributions of the three images that comprise the asterism can be considered as a proof of an astronomical event, such as stellar explosion, or can be used to conclude that the star is a close binary system variable. Such information can be extremely valuable for the specialists in the field, because of the age of the astrographic plates themself.

The legacy astronomical data is a subject of various research both towards reduction [12,14], and even celestial events detection [2]. More recent works also investigate the digitalization of the Carte du Ciel data and its calibration [9], and search for binary and multiple stars by combining data from Carte du Ciel and Gaia catalogues [10].

The goal of our research is to develop image processing and pattern recognition methods and algorithms that will aid the automatic data extraction from digitized Carte du Ciel astrographic maps [7]. As part of our effort, in this paper we present the design and implementation of a convolutional neural network (CNN) [8] for asterism classification in the scanned Carte du Ciel astrographic maps.

Even though well-known frameworks such as PyTorch and open-source libraries such as TensorFlow (see for example [13]) are commonly adopted in similar researches, an important part of our study is to implement custom framework for machine learning via CNN without the usage of a third-party libraries. However, we have used a PyTorch implementation to compare the results of our CNN implementation.

2 CNN for Asterisms Classification

Convolutional neural network (abbreviated CNN or ConvNet) [8] is a class of deep neural networks that are designed to learn pattern features directly from the training data with a contrast to the fully connected neural networks that usually learn patterns from feature vectors that are extracted in preliminary steps [4]. CNNs are considered appropriate for machine learning and classification of complex data such as digital images [3] because of their ability to learn the features that are required to classify patterns directly from the input images.

The ability of CNNs to learn and classify directly from raw images makes them extremely appropriate for our goal to classify asterisms into a predefined number of classes. The scanned astrographic maps in our data set are relatively big images with width of 8750 and height of 8926 pixels. A single astrographic map can contain by a very rough estimation approximately at least 4056 images of asterisms that can be segmented. Note that there are also asterisms which are degraded by various noise and coordinate system that is contained inside the maps, or are too faint to be detected, and they are not subject of interest for the research, because they cannot be analyzed to extract astronomical data. From this point of view, image preprocessing algorithms applied on asterisms in order to extract feature vectors can be quite expensive.

Apart from the above considerations, a major part of our research is focused on the implementation of the CNN. Although our software uses traditional methods and mechanisms similar to those in well-known machine learning platforms and libraries as PyTorch and TensorFlow [13], it does not use any external libraries to create the classification model. Because of the features of our data, mentioned above, we focus on effectiveness, and for that reason our implementation uses C++ programming language, instead of commonly used systems for technical computing such as Matlab or Mathematica, or popular scripting languages such as Python. Something more, currently our software is able to perform one epoch of training on a dataset of 10^4 normalised grayscale images, 28×28 pixels, tiff format, on a standard personal computer configuration in under two minutes. With this performance and for our experiment, we did not need to add more complexity by implementing GPU acceleration.

2.1 Proposed Network Structure

The CNN proposed in this paper is composed by an input, one or more convolutional layers, a set of fully connected layers, and an output layer (see Fig. 2).

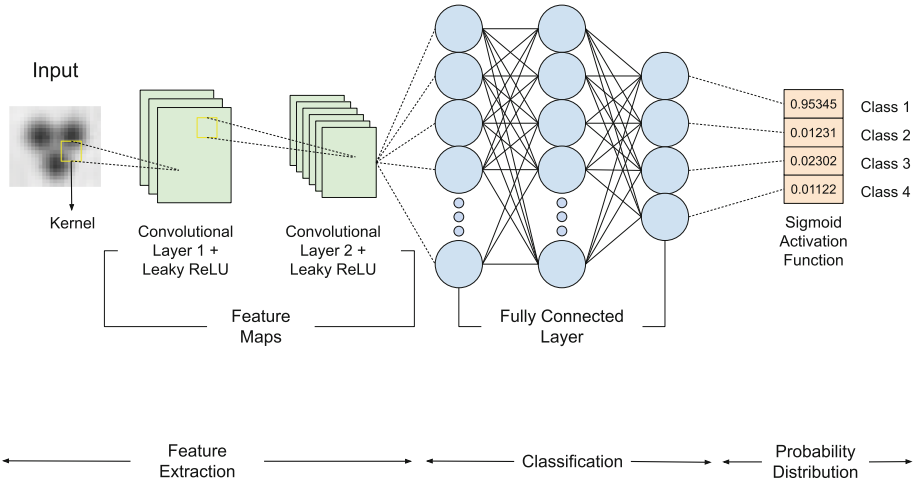


Fig. 2. Topology of the CNN used in our experiment.

Our model allows different number of convolutional layers, and different number and configuration of fully connected layers.

The *input* of the CNN is organized in the form of tensors that store image values [15]. Since a tensor of order K over the field of real numbers $\mathbb{R}$ is denoted

$$\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_K}, \tag{1}$$

a grayscale image with resolution $N \times M$ can be included in a tensor of second order

$$\mathcal{I} \in \mathbb{R}^{N \times M}. \tag{2}$$

Therefore, the tensor of order 3 that can represent the input of the CNN can be written for L number of input images:

$$\mathcal{A} \in \mathbb{R}^{N \times M \times L}. \tag{3}$$

2.2 Convolutional Layer

A *convolutional layer* performs spatial convolution. Convolution in the spatial domain (image domain) is the sum of products between a digital image f and a filter kernel ω with size $n \times m$ (see [3]):

$$(w \star f)(x, y) = \sum_{s=(1-m)/2}^{(m-1)/2} \sum_{t=(1-n)/2}^{(n-1)/2} \omega(s, t) f(x - s, y - t). \tag{4}$$

On Fig. 2, in the feature extraction section, the convolution is shown using the *receptive field* depicted as a yellow square, where the kernel is the matrix of weights that lies in the neighborhood defined by it. The spatial step with which

the receptive field is moved over the image is the *stride*, or in other words it determines the number of pixels the filter moves over the input matrix. A stride value of 1 means that the filter moves over the input image one pixel at a time, while a stride of 2 means that the filter moves over the image two pixels at a time. CNNs use strides that are greater than one for data reduction, and is also used in the process of subsampling or *pooling* that in many architectures is used to achieve invariance to translation. In our current model there is no pooling layer implemented because of the nature of our custom data set *Carte du Ciel Asterism 1)* CDCA1 that we use to train the network (see Sect. 3).

Besides the stride, padding is used by CNN to increase the processing region of the network. In a CNN, a kernel is a filter that moves across an image, scanning each pixel and transforming the data into a smaller or larger format. By adding padding to the image frame, the kernel is provided with additional space to cover the image, which aids in the processing of the image. This allows the CNN to analyze the image more accurately, resulting in better performance.

Our implementation allows multiple convolutional layers, however in most of our experiments we have concluded that two convolutional layers result in better feature extraction and higher accuracy for data that is new for the model. Initially the layer's filter ω is initialised with random coefficients. The convolution process also depends on a set of predefined parameters: image dimensions, ω size n and m, padding, stride, rate of convergence, and bias.

Image dimensions are the image height N and width M, as well as image depth, that in our case is determined by the fact that we use grayscale images with each pixel having an integer value in the closed interval $[0, 255]$.

Kernel size is represented by the dimensions of ω, that are n and m, and by filter repetition and depth.

Padding, which we denote by *pad*, is a single value, representing how many pixels to add on each side of the image. It is used to prevent loss of data around the edges or to enable the usage of larger filters.

Stride, denoted *str*, is also a single value, representing how much the convolutional filter will move on each iteration.

Rate of convergence r_{conv} is a decimal value, representing how quickly the algorithm reaches a solution that we can define as an optimal regarding a given criterion.

Bias b is a decimal value, representing the shift of the output. It is used to shift the activation function result of each neuron in order to clear out any potential offsets.

The convolutional layer has forward and backwards propagation leveraged by the *leaky rectified linear* (Leaky ReLU) [11]:

$$R(z) = \max(\alpha z, z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha z, & \text{otherwise} \end{cases} \tag{5}$$

The first derivative of (5) is then:

$$R'(z) = \begin{cases} 1, & \text{if } z > 0 \\ \alpha, & \text{if } z < 0 \\ \text{indeterminate} & \text{if } z = 0 \end{cases} \tag{6}$$

The constant α determines the slope of the function for negative inputs, and usually $\alpha = 0.01$ or $\alpha = 0.2$. The reason for setting the constant α is to resolve the common problem with the "dying" neurons when the input of the standard rectified linear activation function is a negative number, leading to a zero neuron output.

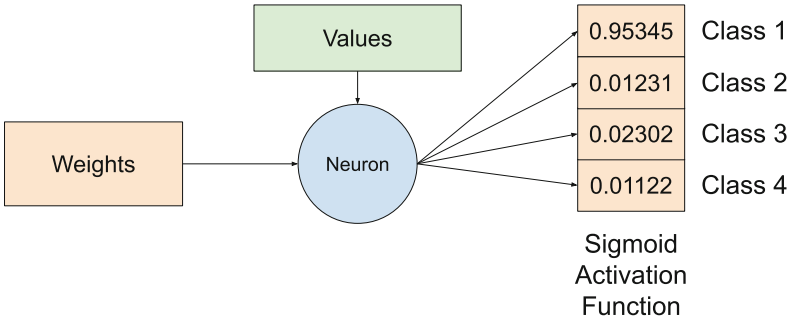


Fig. 3. Scheme of the sigmoid neuron used to build fully connected layers of the CNN.

2.3 Fully Connected Layer

Each fully connected layer l of the CNN is composed by a set of sigmoid neurons, where the ith neuron (Fig. 3) can be represented (see for example [3]

$$z_i(l) = \sum_{j=1}^{s_{l-1}} w_{ij} a_i(l-1) + b_i(l), \tag{7}$$

where s_{l-1} denotes the number of neurons in the previous layer, w_{ij} are the weights, and $b_i(l)$ is the bias weight. $a_i(l) = h(z_i(l))$ defines the output or activation value of the neuron, and in our case the activation function h is the *sigmoid function*:

$$h(z) = \frac{1}{1 + e^{-z}}. \tag{8}$$

Our implementation supports multiple fully connected layers, and for each layer l, it allows the definition of the number of neurons s_l. The hyperparameters that are configurable are: number of input layers, sublayers (hidden layers), number of output classes, bias, adam, and rate of convergence.

The *input layer* is a single number, representing the number of input layers, coming out of the convolutional layer.

The *sublayer* is a vector, representing the topology of the fully connected layer.

The number of output classes is a single number, representing how many output classes are available for the specific dataset. Latter we will show that for our experiments with the MNIST data set [1] it is set to 10, and in our experiments with the asterisms data set, it is set to 4.

The bias is a decimal value, representing the shift of the output.

The *adam* hyperparameter is a boolean value, representing whether to use Adam or SGD as optimization algorithms.

A layer l from the fully-connected section of the CNN calculates class scores for each iteration. It uses the sigmoid activation function given in (8), and since it is uncommon to adopt more than one activation function in a CNN, in our approach we do not need to apply Softmax or cross-entropy loss functions (see [4]). The result of the sigmoid activation function is directly the probability distribution of classes in the closed interval $[0, 1]$.

2.4 Training the Model

The training of the defined model follows the standard algorithm for training of a CNN (see for example [3] and [4]):

1. Input image data in the form of tensor $\mathcal{A}$ defined in (3).
2. Forward pass for each neuron in each feature map.
3. Back propagation for each neuron in each feature map.
4. Weights and bias weight update for each feature map.

A complete run through the entire data set of the above four steps defines one *epoch* of training.

The training algorithm accepts the number of epochs to be performed, logging options and if a validation on each epoch is necessary. During training the parameters of the network are changed in order to get as high accuracy as possible. Validation is usually performed after each epoch with new, unseen data to simulate the testing phase with the weights calculated after each epoch. It gives early warning if the model performs well when faced with new, unseen data, during the validation, the weights are not updated.

Finding the appropriate set of parameters of a neural network that can reduce the cost of the selected cost function that evaluates the performance is a subject of optimization of the training stage. We have implemented two standard optimization algorithms to optimize the process of learning: stochastic gradient descent (SGD), and adaptive moments (Adam) [4].

The SGD algorithm is based on a selection of a relatively small set of random samples that are used to estimate the gradient in each iteration. Then the algorithm follows the gradient downhill.

Adam is an adaptive learning rate optimization algorithm that estimates the gradient based on the first-order, and uncentered second-order moments form the selected samples.

Each epoch loops through all images one by one, forwards it through the layers, calculates the loss, calculates the accuracy and lastly backwards the image through the layers to update the weights.

We adopt *mean squared error* (MSE) loss function

$$MSE = \frac{1}{L}\sum_{i=1}^{L}(y_i - \widehat{y}_i), \tag{9}$$

where L is the number of input images contained in $\mathcal{A}$, y_i is the actual value of the target variable for the ith image, and $\widehat{y}_j$ is the predicted value of the target variable for the ith image. Apparently, the lower the MSE value is, the better performance of the neural network is.

3 CDCA1 and Experiment Results

Table 1. Number of training epochs, training and testing accuracies in percentage achieved with PyTorch and our CNN implementation for both datasets.

	MNIST			CDCA1		
	Epoch	Train	Test	Epoch	Train	Test
PyTorch	5	95.5	95.4	5	74	74
Our library	5	88	83	5	83	73

We have tested our implementation of CNN using two different data set:

1. MNIST data set [1].
2. The first version of our custom Carte du Ciel Asterisms data set (CDCA1).

In both cases we have adopted the CNN architecture that is given in Fig. 2:

1. Two convolutional layers leveraged by the Leaky ReLU function (5).
2. Fully connected layers:
   - the first fully connected layer, composed by 72 sigmoid neurons;
   - the hidden layer, also composed by 72 sigmoid neurons.
3. Output layer that gives the probability for classification in each of the predefined number of classes that determines the number of output neurons.

Also we have compared and verified our implementation by performing the same experiment using an analogous CNN implemented using Python and

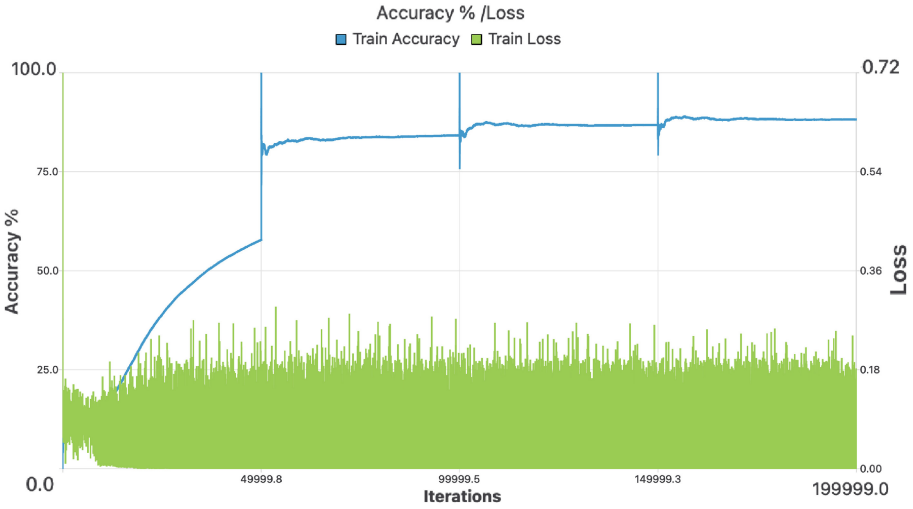


Fig. 4. Train accuracy and train loss of our CNN for the MNIST data set.

PyTorch framework (see Table 1. The main difference between our implementation and PyTorch based implementation is that the latter has also a pooling layer.

We have used the MNIST data set to verify our implementation of CNN prior to our experiments with the asterisms extracted from the Carte du Ciel astrographic maps. The MNIST database contains images of handwritten digits, so the number of output neurons in this case is 10. The dataset consists of 5×10^4 training images and 10^4 testing images, from these 5×10^4 we use 10^4 for validating after the completion of each epoch. However, it should be pointed out that the MNIST dataset is a comprehensive dataset, carefully verified and improved throughout the years of its existence and the images are equally distributed in the all 10 classes. Our CNN reaches 88% accuracy after 5 epoch of training (see Fig. 4).

As expected, for the MNIST data set the PyTorch implementation performs better than the proposed CNN implementation (see Table 1), and the reason for this results is the pooling layer in the PyTorch CNN. However, the experiments with MNIST data set are performed to verify and test the implementation of the proposed software solution.

For the purpose of the experiments with the Carte du Ciel astrographic maps data we have developed the first version of our data set CDCA1 (Carte du Ciel Asterism) that contains segmented images of asterisms from the scanned data. Initially, we have segmented manually more than 1000 asterisms to form the base of the data set. Since this number of input images is quite insufficient to train and test our CNN, we have developed an extra module in our software that generates the CDCA1 by augmenting the base images using the following operations:

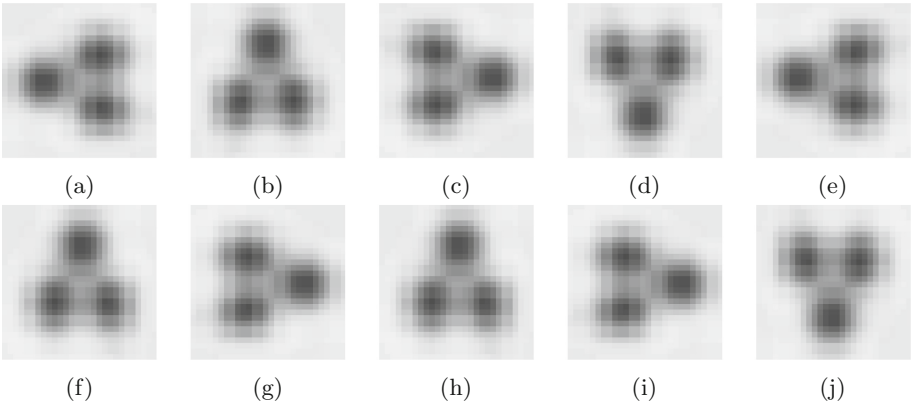


Fig. 5. Transform the original image: (a) rotate by 90°; (b) rotate by 180°; (c) rotate by 270°; (d) horizontal flip; (e) horizontal flip and rotate by 90°; (f) horizontal flip and rotate by 180°; (g) horizontal flip and rotate by 270°; (h) vertical flip; (i) vertical flip and rotate by 90°; (j) vertical flip and rotate by 180°.

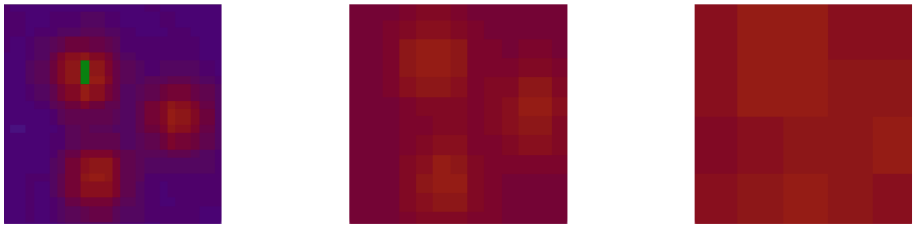
- resize images to 28×28 pixels;
- normalize images and set all images to grayscale format;
- apply geometrical transformations of rotation, horizontal and vertical flip, and their combinations, given in Fig. 5.

In this way, the resulting CDCA1 dataset consists of more than 14000 asterism images.

Fig. 6. The four classes of asterisms in which the CNN classifies data in CDCA1.

We define four classes of asterisms in the dataset CDCA1 (their representatives are given in Fig. 6):

- class C_1 represents asterisms of bright stars whose three images are merged together in a single connected component;
- class C_2 represents the most common case of asterisms composed by three stars images that have relatively equal Gaussian distributions;
- class C_3 represents asterisms composed by two bright and one fainter image;



(a) (b) (c)

Fig. 7. Asterism image heatmap: (a) input of the CNN; (b) after the first convolutional layer; (c) after the second convolutional layer.

– class C_4 represents asterisms composed by two faint and one brighter image.

The CNN achieves 83% accuracy on the CDCA1 dataset after 5 epochs of training (see Fig. 8). During the testing phase, the accuracy reaches 74%. The lower testing accuracy is normal in this case because a single epoch is performed, and the testing data is unknown for the CNN. The result of the convolutional layers is illustrated on Fig. 7, where are given the heatmaps: of the input of the CNN; after the first convolutional layer; and after the second convolutional layer.

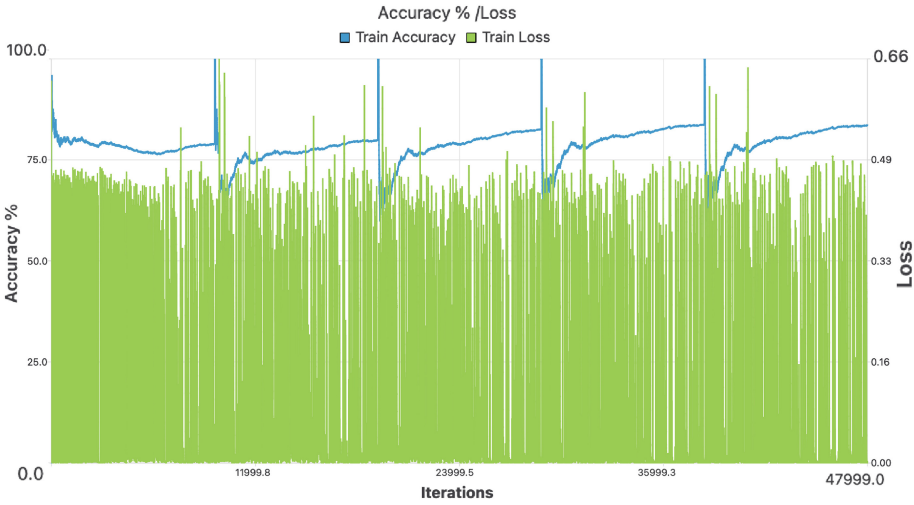


Fig. 8. Train accuracy and train loss of our CNN for the proposed CDCA1 data set.

As it can be seen from the results in Table 1 the performance of the PyTorch implementation and our CNN implementation are extremely close, even though our implementation does not provide a pooling layer. As mentioned above, it can be explained by the nature of the CDCA1 data set, and also by the geometrical distortions that we have applied in order to augment artificially the data set.

4 Conclusion

In this paper we present our implementation of a convolutional neural network (CNN) whose purpose is the automatic classification of asterisms in digitized Carte du Ciel astrographic maps. Our implementation is written in the C++ programming language, and the CNN is implemented without any use of third-party library. The software is tested and verified using the popular MNIST dataset. In order to perform our experiments on the astrographic maps data, we have created our custom CDCA1 dataset, that is composed by manually segmented asterisms, and is augmented using geometrical transformations in order to achieve a number of images that is sufficient to train and test CNN.

The results on CDCA1 show 83% accuracy after 5 epochs of training and 73% accuracy in the testing phase. The lower accuracy during the testing phase can be explained with overfitting that occurs when the model becomes too close to the data point of the training section of the data set. It is a natural result of the augmentation that is used to generate CDCA1. However, these results clearly show that the selected approach is appropriate for the automatic classification of asterisms in the examined data.

As future work, we will implement an option to add a pooling layer to the topology of our CNN. We will also integrate our CNN model with a software for automatic asterisms segmentation from Carte du Ciel astrographic maps [7], which currently is under process of development. The latter will also result in the second version of our dataset CDCA2 that will not involve artificial augmentation, and will contain unique segmented asterisms images. The latter will allow us to perform additional experiments for fine tuning of the CNN hyperparameters.

References

1. Deng, L.: The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. **29**(6), 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477
2. Fresneau, A., Argyle, R.W., Marino, G., Messina, S.: Potential of astrographic plates for stellar flare detection. Astron. J. **121**(1), 517–524 (2001). https://doi.org/10.1002/nav.3800020109
3. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Pearson, New York (2018)
4. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
5. Jones, D.: The scientific value of the Carte du Ciel. Astron. Geophys. **41**(5), 5.16–5.20 (2000). https://doi.org/10.1046/j.1468-4004.2000.41516.x
6. Krizhevsky, A., Sutskever, I., E.Hinton, G.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
7. Laskov, L.M., Tsvetkov, M.: Data extraction form Carte du Ciel tripple images. Serdica J. Comput. **7**(4), 317–332 (2013)
8. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 253–256 (2010). https://doi.org/10.1109/ISCAS.2010.5537907

9. Lehtinen, K., et al.: Digitization and astrometric calibration of Carte du Ciel photographic plates with Gaia DR1. Astron. Astrophys. **616**, A185 (2018). https://doi.org/10.1051/0004-6361/201832662
10. Lehtinen, K., et al.: Carte du Ciel and Gaia - i. astrometry. Astron. Astrophys. **671**, A16 (2023). https://doi.org/10.1051/0004-6361/202142929
11. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: ICML Workshop on Deep Learning for Audio, Speech, and Language Processing, pp. 1–6 (2013)
12. Ortiz-Gil, A., Hiesgen, M., Brosche, P.: A new approach to the reduction of Carte du Ciel plates. Astron. Astrophys. Suppl. Ser. **128**(3), 621–630 (1998). https://doi.org/10.1051/aas:1998168
13. Prakash, K.B., Kanagachidambaresan, G.R. (eds.): Programming with TensorFlow. EAI/Springer Innovations in Communication and Computing, Springer Cham (2022). https://doi.org/10.1007/978-3-030-57077-4
14. Urban, S.E.: New reductions of the astrographic catalogue: high accuracy, early epoch positions for proper motion studies. Int. Astron. Union Colloq. **165**, 493–498 (1997). https://doi.org/10.1017/S025292110004700X
15. Vasilescu, M.A.O., Terzopoulos, D.: Multilinear (tensor) image synthesis, analysis, and recognition [exploratory DSP]. IEEE Signal Process. Mag. **24**(6), 118–123 (2007). https://doi.org/10.1109/MSP.2007.906024