The Journal of the Astronautical Sciences (2020) 67: 1063–1091
https://doi.org/10.1007/s40295-019-00208-w

Space Objects Classification via Light-Curve Measurements Using Deep Convolutional Neural Networks

Richard Linares[1] · **Roberto Furfaro**[2] · **Vishnu Reddy**[3]

Published online: 12 March 2020

Abstract

This work presents a data-driven method for the classification of light curve measurements of Space Objects (SOs) based on a deep learning approach. Here, we design, train, and validate a Convolutional Neural Network (CNN) capable of learning to classify SOs from collected light-curve measurements. The proposed methodology relies on a physics-based model capable of accurately representing SO reflected light as a function of time, size, shape, and state of motion. The model generates thousands of light-curves per selected class of SO, which are employed to train a deep CNN to learn the functional relationship between light-curves and SO classes. Additionally, a deep CNN is trained using real SO light-curves to evaluate the performance on real data, but limited training set. The CNNs are compared with more conventional machine learning techniques (bagged trees, support vector machines) and are shown to outperform such methods, especially when trained on real data.

Keywords Deep learning · Convolutional neural network · Light curve processing · Space situational awareness

✉ Richard Linares
linaresr@mit.edu

Roberto Furfaro
robertof@email.arizona.edu

Vishnu Reddy
reddy@lpl.arizona.edu

[1] Massachusetts Institute of Technology, Cambridge, MA 02139, USA

[2] Department of Systems and Industrial Engineering, Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson, AZ, USA

[3] Lunar and Planetary Lab, University of Arizona, Tucson, AZ, USA

Introduction

Motivated by the U.S. Air Force mission to control, protects, and maintain access to space, Space Situational Awareness (SSA) has recently become an important research topic [1]. Researchers, and operational engineers, rely on a large amount of tracking data that can be processed to identify, characterize, and understand the intention of Space Objects (SO). The SO catalog maintained by the U.S. Air Force currently includes upward of 19,870 SO on orbit [2], with 2,434 of such objects being actively controlled and operated. Researchers working on SSA are interested in providing a detailed understanding of SO population behavior, which must go beyond the currently SO catalog comprising simplified SO characteristics, such as solar radiation pressure and drag coefficients. To provide a more realistic and reliable understanding of the SO dynamics, future catalogs must include detailed SO characteristics (e.g., shape and state of motion). The latter can be employed in dynamical propagation models to predict SO trajectory and behavior accurately.

Developing a detailed understanding of the SO population is a fundamental goal of SSA. Such a simplified description limits the dynamic propagation model used for predicting the state of motion of SO to models that assume cannonball shapes and generic surface properties. The future SO catalog and SSA systems will have to be capable of building a detailed picture of SO characteristics. Traditional measurement sources for SO tracking, such as radar and optical, provide information on SO characteristics.

Optical sensors are generally employed to track near-geosynchronous SO. Such sensors provide both astrometric and photometric measurements. Consequently, SO properties can be estimated from astrometry (e.g., trajectories) and photometric data (e.g., shape and state of motion). More specifically, light-curves, i.e., flux of photons across a wavelength reflected by the SO and collected by optical sensors, play an important role in determining the SO attitude and state of motion. Indeed, attitude estimation and extraction of other characteristics using light curve data have been demonstrated in Ref. [3, 10–13].

Traditional measurement sources for SO tracking (e.g. radar and/or optical measurements) have been shown to be sensitive to shape [3, 4], attitude [3, 5, 6], angular velocity [7], and surface parameters [8, 9]. A literature review shows that recent advancements have been made to estimate SO properties. Such techniques heavily rely on estimation theory and include the development of multiple model [3, 14], nonlinear state estimation [5–7], and full Bayesian inversion [15] approaches for SO characterization. Although grounded in a solid theoretical background, the above-mentioned methods tend to be computationally expensive. New techniques are sought that can provide a higher degree of accuracy, computational efficiency, and reliability.

Generally, classifying SO is a challenging task. State-of-the-art methods rely on well established physical models that are embedded in an inversion scheme capable of processing the data and estimate the model parameters. For example, Reference 3 used a Multiple Model Adaptive Estimation classification approach to model the dynamics and physics, estimate relevant parameters, and finally classify SOs. Although such a method is one of the most promising available in the literature,

the inversion process requires the estimation of a large number of parameters. As a result, the computational burden is significant and may not be practical for a catalog comprising a large number of objects. Here, we are interested in exploring a data-driven classification approach that employs both simulated and real-data to learn the functional relationship between observed light-curves and SO class.

Recent advancements in machine learning have included deep learning as a critical breakthrough technology. Indeed, deep learning methods [16] have shown groundbreaking results across a large number of domains. Deep networks are neural networks that comprise more than hidden layers of neurons in their architecture. In such multi-layer neuronal arrangement, deep learning approaches are designed to mimic the function of the brain by learning nonlinear hierarchical features from data that build in abstraction [17]. The latter enabled a higher level of accuracy in typical classification tasks. In this paper, we explore deep learning methods to classify SOs trained on their simulated and real data. More specifically, we investigate Convolutional Neural Networks (with max-pooling and dropout) [17] for supervised classification of SO observational data. Here, we demonstrate the design of CNN architectures trained both on simulated and real data, and demonstrate the effectiveness of the proposed methodology in classifying SOs.

The Convolutions Neural Network (CNN) have achieved remarkable performance on image processing tasks. Examples include 1) object classification [18], 2) scene classification [19], and 3) video classification [20]. Importantly, the key enabling factor for the success of CNN is the development of techniques that can optimize large scale networks, comprising tens of millions of parameters, as well as massive labeled datasets. Inspired by these results, this paper studies the classification of observational data generated by both simulated and real dynamical systems. The dynamical system investigated here is the rotational dynamics of SOs. The physical attributes of the SOs, such as shape and mass distribution, are also included in the classification process. The challenging aspect of the application of CNNs to physical dynamical systems is the generation of labeled training data. In a dual-fold fashion, we first use physical models to simulate observations by sampling randomly from a distribution of physical attributes and dynamic states. Light curve measurements are used as inputs, and classes of the SOs are used as outputs for training the CNN approach. The 1D-CNN then learns convolutional kernels that look for characteristic features in the light curve data. As opposed to manually specified features, the features are adaptively learned, given the training data. Subsequently, available and labeled real light-curves are employed to train a specified CNN architecture directly.

This work compares the CNN with two machine learning techniques that represent somewhat the state of the art of automatic classification techniques. The first method is called Bagged Trees, and it is an ensemble method. The idea behind ensemble methods is to combine many weak learners into a highly accurate single ensemble algorithm. Thus, we can consider a classification ensemble as a machine learning model comprising a weighted combination of many individual classification algorithms. This work considers a type of ensemble algorithm called *Bagging* (where *Bag* stands for *Bootstrap aggregation* [21]) combined with a basic decision tree algorithm [22]. The bagging process on a decision tree works by generating many bootstrap replicas of the training dataset and then growing decision trees on such replicas. Each

of the bootstrap replicas are obtained by randomly choosing N observations and considering N replacements (where N is the training set size). The method works by simply training each of the individual weak learners on resampled versions of the training set. The overall response of the model is obtained by average prediction over the ensemble of individual learners. The second set of techniques is the well-known Support Vector Machines (SVM, [23]) adapted for the multi-class case. SVM generally classify data by finding the hyperplane that separates two classes by the largest margin. Non-linear transformations are generally employed when the classes are non-separable by a simple (linear) hyperplane. In this case, a variety of kernels are available to execute the transformation. The data-driven deep learning approach is compared with more conventional machine learning techniques (e.g., random forest [24] and SVM [23]) to show that hierarchical feature learning is the key to building successful discriminative models.

The organization of this paper is as follows. First, a section reviewing the light curve and SO dynamics model used for this work is provided. Next, the generation of training data and real data are discussed with an outline of the simulated labeled data approach and the model used for generating attitude control profiles. Then the theory behind the CNN is provided, followed by the t-SNE technique. Then numerical results are given. Finally, concluding remarks are provided on the performance of the proposed CNN-based approach.

Light Curve and Dynamics of Space Objects Modeling

Light Curve Modeling

There are several models used for simulating light curve measurements in literature, and Ref. 14 provides a good summary of the most popular ones adopted for space objects (SO) applications. These models differ in the physics that they represent and their level of complexity, but for SOs applications, the ability to model specular reflection and complex shapes while converting energy is desirable. The Ashikhmin-Shirley [25] (AS) model has all the desirable properties while producing realistic SOs light curve. This model is based on the bidirectional reflectance distribution function (BRDF), which models light distribution scattered from the surface due to the incident light. The BRDF at any point on the surface is a function of two directions, the direction from which the light source originates and the direction from which the scattered light leaves the observed surface. The model in Ref. 25 decomposes the BRDF into a specular component and a diffuse component. The two terms sum to give the total BRDF:

$$f_r = (d R_d + s R_s) \tag{1}$$

which depends on the diffuse bidirectional reflectance (R_d) and the specular bidirectional reflectance (R_s) and the fraction of each to the total (d and s respectively where $d + s = 1$). Each facet contributes independently to the brightness, and total brightness is the sum over each facet's contribution. The diffuse component represents the light that is scattered equally in all directions (Lambertian), and the specular

component represents the light that is reflected strongly in some direction (mirror-like). Reference 25 develops a model for continuous arbitrary surfaces but simplifies for flat surfaces. This simplified model is employed in this work as shape models are considered to consist of a finite number of flat facets. Therefore the total observed brightness of an object becomes the sum of the contribution from each facet.

In each model, however, $c = \mathbf{V}^T\mathbf{H}$, ρ is the diffuse reflectance ($0 \leq \rho \leq 1$), and F_0 is the specular reflectance of the surface at normal incidence ($0 \leq F_0 \leq 1$). To be used as a prediction tool for brightness and radiation pressure calculations, an important aspect of the BRDF is energy conservation. For energy to be conserved, the integral of the BRDF times $\cos(\theta_r)$ over all solid angles in the hemisphere with $\theta_r \leq 90$ needs to be less than unity, with

$$\int_0^{2\pi}\int_0^{\pi/2} f_r \cos(\theta_r)\sin(\theta_r)\, d\theta_r d\phi = R_d + R_s \tag{2}$$

For the BRDF given in Eq. 1, this corresponds to constant values of $R_d = \rho d$ and $R_s = sF_0$. The remaining energy not reflected by the surface is either transmitted or absorbed. In this paper, it is assumed the transmitted energy is zero. The diffuse bidirectional reflectance is then calculated as follows:

$$R_d = \frac{28\rho}{23\pi}(1 - sF_0)\left(1 - \left(1 - \frac{\mathbf{N}^T\mathbf{L}}{2}\right)^5\right)\cdot \left(1 - \left(1 - \frac{\mathbf{N}^T\mathbf{V}}{2}\right)^5\right) \tag{3}$$

where

$$F = F_0 + \left(\frac{1}{s} - F_0\right)(1 - c)^5 \tag{4}$$

The vectors $\mathbf{L}(t_i)$ and $\mathbf{V}(t_i)$ denote the unit vector from the SO to the Sun and the unit vector from the SO to the observer, respectively, and together they define the observation geometry (shown in Fig. 1). In addition to d, ρ, and F_0, the Ashikhmin-Shirley BRDF has two exponential factors (n_u, n_v) that define the reflectance properties of each surface. The Ashikhmin-Shirley diffuse and specular reflectivities are not constant but rather complicated functions of illumination angle, exponential factor, and the diffuse and specular reflectances. In all cases, however, $R_d + R_s \leq 1$, so energy is conserved. The parameters of the Phong model that dictate the directional (local horizontal or vertical) distribution of the specular terms are n_u and n_v. The specular bidirectional reflectance for the AS model is given by

$$R_s = \frac{F\sqrt{(n_u + 1)(n_v + 1)}}{8c\pi \max\left[\mathbf{N}^T\mathbf{L}, \mathbf{NV}\right]}(\cos(\alpha))^{\gamma} \tag{5}$$

where $\gamma = n_u \cos^2(\beta) + n_v \sin^2(\beta)$.

The apparent magnitude of an SO is the result of sunlight reflecting off of its surfaces along the line-of-sight to an observer. First, the fraction of visible sunlight

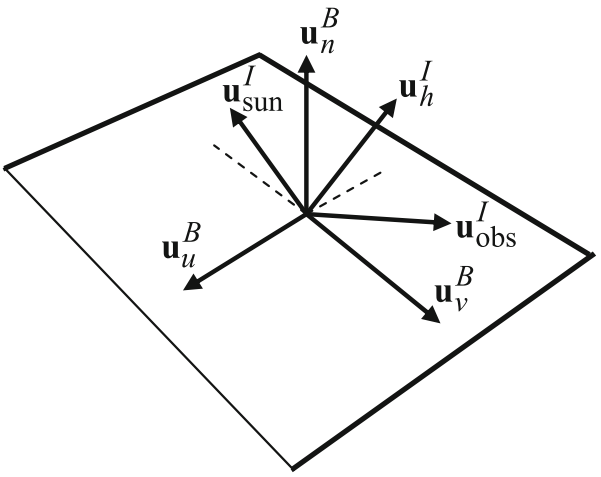


Fig. 1 Reflection geometry

that strikes an object (and is not absorbed) is computed by

$$F_{\text{sun}}(i) = C_{\text{sun,vis}} \left(\mathbf{u}_n^I(i) \cdot \mathbf{u}_{\text{sun}}^I \right) \tag{6}$$

Where i denotes the i^{th} facet of the SOs. $C_{\text{sun,vis}} = 1062\ \text{W/m}^2$ is the power per square meter impinging on a given object due to visible light striking the surface. If either the angle between the surface normal and the observer's direction or the angle between the surface normal and Sun direction is greater than $\pi/2$ then there is no light reflected toward the observer. If this is the case then the fraction of visible light is set to $F_{\text{sun}}(i) = 0$. Next, the fraction of sunlight that strikes an object that is reflected must be computed:

$$F_{\text{obs}}(i) = \frac{F_{\text{sun}}(i)\rho_{\text{total}}(i)\mathcal{A}(i)\left(\mathbf{u}_n^I(i) \cdot \mathbf{u}_{\text{obs}}^I\right)}{\|\mathbf{d}^I\|^2} \tag{7}$$

The reflected light of each facet is now used to compute the total photon flux, which is measured by an observer:

$$\tilde{F} = \left[\sum_{i=1}^{N} F_{\text{obs}}(i) \right] + v_{\text{CDD}} \tag{8}$$

where v_{CDD} is the measurement noise associated with flux measured by a Charge Coupled Device (CCD) sensor. The total photon flux is then used to compute the apparent brightness magnitude

$$m_{\text{app}} = -26.7 - 2.5 \log_{10} \left| \frac{\tilde{F}}{C_{\text{sun,vis}}} \right| \tag{9}$$

where -26.7 is the apparent magnitude of the Sun.

Attitude Dynamics of Space Objects

A number of parameterizations exist to specify attitude, including Euler angles, quaternions, and Rodrigues parameters [26]. This paper uses the quaternion, which is based on the Euler angle/axis parameterization. The quaternion is defined as $\mathbf{q} \equiv [\boldsymbol{\varrho}^T \quad q_4]^T$ with $\boldsymbol{\varrho} = \hat{\mathbf{e}} \sin(\nu/2)$, and $q_4 = \cos(\nu/2)$, where $\hat{\mathbf{e}}$ and ν are the Euler axis of rotation and rotation angle, respectively. Clearly, the quaternion must satisfy a unit norm constraint, $\mathbf{q}^T\mathbf{q} = 1$. In terms of the quaternion, the attitude matrix is given by

$$A(\mathbf{q}) = \Xi^T(\mathbf{q})\Psi(\mathbf{q}) \tag{10}$$

where

$$\Xi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} + [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{11a}$$

$$\Psi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_{3\times 3} - [\boldsymbol{\varrho}\times] \\ -\boldsymbol{\varrho}^T \end{bmatrix} \tag{11b}$$

with

$$[\mathbf{g}\times] \equiv \begin{bmatrix} 0 & -g_3 & g_2 \\ g_3 & 0 & -g_1 \\ -g_2 & g_1 & 0 \end{bmatrix} \tag{12}$$

for any general 3×1 vector $\mathbf{g}$ defined such that $[\mathbf{g}\times]\mathbf{b} = \mathbf{g} \times \mathbf{b}$.

The rotational dynamics are given by the coupled first order differential equations:

$$\dot{\mathbf{q}}_I^B = \frac{1}{2}\Xi(\mathbf{q}_I^B)\boldsymbol{\omega}_{B/I}^B \tag{13a}$$

$$\dot{\boldsymbol{\omega}}_{B/I}^B = J_{\mathrm{SO}}^{-1}\left(\mathbf{T} + \mathbf{T}_{\mathrm{srp}}^B - \left[\boldsymbol{\omega}_{B/I}^B\times\right] J_{\mathrm{SO}}\boldsymbol{\omega}_{B/I}^B\right) \tag{13b}$$

where $\boldsymbol{\omega}_{B/I}^B$ is the angular velocity of the SO with respect to the inertial frame, expressed in body coordinates, and J_{SO} is the inertia matrix of the SO. The vectors $\mathbf{T}_{\mathrm{srp}}^B$ and $\mathbf{T}$ are the net torques acting on the SO due to SRP expressed in body coordinates and the control torque, respectively. In this work, we assume no Solar Radiation Pressure (SRP) torque but include control torque for the SO classes with control. The net torques are $\mathbf{T} = \mathbf{T}_{\mathrm{control}}$.

Training Set Generation

Simulating Labeled Training Data

For the simulated case, the labeled training data samples are generated using the light curve model discussed above. The parameters required to define the AS light curve model are sampled. We considered four categories, i.e., fragments, rocket bodies, regular polygon prisms, and rectangular cuboids. The SO parameter models associated with shape and surface are randomly generated out of a uniform distribution. Importantly, the regular polygon prisms are then further divided into equilateral triangular

prisms, square prisms, and regular hexagonal prisms. The regular polygon prisms are prisms whose ends (i.e. top and bottom) are regular shapes. The shape of a regular polygon prism is defined by the number of sides n, side length s, and height h. These parameters are sampled from the following distribution

$$h_{\text{regular}} = (h_{\min} + 0.01) + (h_{\max} - h_{\min} - 0.01)\,\mathcal{U}[0, 1] \tag{14a}$$

$$s_{\text{regular}} = (s_{\min} + 0.01) + (s_{\max} - s_{\min} - 0.01)\,\mathcal{U}[0, 1] \tag{14b}$$

Assuming constant density throughout the shape model, the moment of inertia matrices for each of the regular polygon models are given by

$$J_{\text{triangle}} = m_{\text{SO}} \begin{bmatrix} \frac{s^2}{24} + \frac{h^2}{12} & 0 & 0 \\ 0 & \frac{s^2}{24} + \frac{h^2}{12} & 0 \\ 0 & 0 & \frac{s^2}{12} \end{bmatrix} \tag{15a}$$

$$J_{\text{square}} = m_{\text{SO}} \begin{bmatrix} \frac{s^2}{12} + \frac{h^2}{12} & 0 & 0 \\ 0 & \frac{s^2}{12} + \frac{h^2}{12} & 0 \\ 0 & 0 & \frac{s^2}{6} \end{bmatrix} \tag{15b}$$

$$J_{\text{hexagon}} = m_{\text{SO}} \begin{bmatrix} \frac{5s^2}{24} + \frac{h^2}{12} & 0 & 0 \\ 0 & \frac{5s^2}{24} + \frac{h^2}{12} & 0 \\ 0 & 0 & \frac{5s^2}{24} \end{bmatrix} \tag{15c}$$

The rectangular cuboids are prisms defined by two side lengths s_1 and s_2 as well as the height h. The moment of inertia matrix for the cuboids are given by

$$J_{\text{cuboid}} = \frac{m_{\text{SO}}}{12} \begin{bmatrix} s_2^2 + h^2 & 0 & 0 \\ 0 & s_1^2 + h^2 & 0 \\ 0 & 0 & s_1^2 + s_2^2 \end{bmatrix} \tag{16}$$

The models are generated by sampling side lengths and heights from a uniform distribution on the interval $[0.01, 5]$ m. For the regular polygon prisms, the number of sides is also selected randomly on the interval $[3, 6]$, with all instances of 5 sides being set to 4 as pentagonal prism models are not included. In addition to the model geometry, the material properties also need to be defined. For each model, all facets are assumed to have the following: $R_{\text{spec}} = 0.7$, $R_{\text{diff}} = 0.3$, $\epsilon = 0.5$. The Phong parameters n_u and n_v are each taken to be equal to 1000 for all facets of every model. The mass of the SO is randomly sampled using the following $m_{\text{SO}} = m_{\min} + (m_{\max} - m_{\min})\mathcal{U}[0, 1]$.

The rocket body models are generated using octant triangulation of a sphere as discussed in Ref. [27], which divided the surface of a sphere into N facet normal. Then the rocket body models are generated by connecting two hemisphere ends of

radius r with a cylinder of height l. This model is not exact for all rocket bodies but is close enough to approximate the types of light-curves seen for rocket bodies.

$$\begin{aligned} J_{\text{rocket}} = m_{\text{SO}} \Bigg\{ & \frac{V_{\text{cyl}}}{V_{\text{tot}}} \text{diag}\left[\frac{1}{12}(3r^2+l^2), \frac{1}{12}(3r^2+l^2), \frac{r^2}{2}\right] \\ & + \frac{V_{\text{top}}}{V_{\text{tot}}} \text{diag}\left[\frac{1}{12}(3r^2+l^2), \frac{1}{12}(3r^2+l^2), \frac{r^2}{2}\right] \\ & + \left(\frac{V_{\text{top}}}{V_{\text{tot}}}\left(\frac{l}{2}+\frac{3r}{8}\right) + \frac{V_{\text{cyl}}}{V_{\text{tot}}}\left(\frac{l}{2}-\frac{3r}{8}\right)\right)\left(I_{3\times 3} - \mathbf{e}\mathbf{e}^T\right) \\ & + 2\frac{V_{\text{top}}}{V_{\text{tot}}} r^2 \text{diag}\left[\frac{83}{320}, \frac{83}{320}, \frac{2}{5}\right] \Bigg\} \end{aligned} \tag{17}$$

Where $\mathbf{e} = [0, 0, 1]^T$ and the volume of the top hemisphere is given by $V_{\text{top}} = 2/3\pi r^3$ and it is assumed the bottom volume is $V_{\text{bot}} = V_{\text{top}}$. The volume of the cylinder is given by $V_{\text{cyl}} = \pi r^2 l$ and the total volume is $V_{\text{tot}} = V_{\text{top}} + V_{\text{bot}} + V_{\text{cyl}}$. Finally, the fragment shapes use the cuboid model but with much small aspect ratios than payload shapes. Figure 2 shows the training data generated using the process discussed above where the labels are shown using colored data points.

Attitude Control Profile Modeling

This section follows the simulated attitude control profiles discussed in Ref. [28]. When processing light curve observations, it may not be valid to assume that the

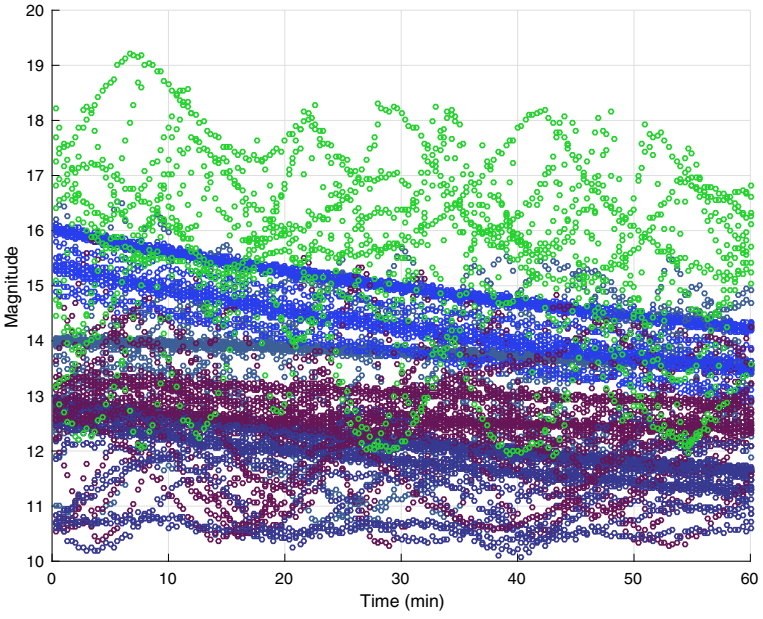


Fig. 2 This figure shows the labeled simulated training data where each label is given a unique color

SO is uncontrolled, and therefore, we must take into account the possibility of controlled attitude states. Determining whether a SO has active control or not may also provide a feature state that may be used for classification. For example, a determination of whether a SO is passive or active can be made based on whether light curve observations indicated that the SO has active attitude control.

In this work, the attitude control is simulated by assuming control profiles. This work uses three possible control profiles, Sun pointing, Nadir pointing, and spin-stabilized. Then for each control profile, a desired angular velocity is determined, which will allow the SO to track the relevant directions. The angular velocity profiles are used to calculate the torque required to follow this profile. This section discusses the attitude control approach used. This work uses a simple model for developing the simulated attitude controllers and for more detail please refer to Ref. [29]. The attitude control is designed to minimize the following error:

$$\mathbf{e} = \boldsymbol{\omega} - \boldsymbol{\omega}_d \tag{18}$$

Differentiating this equation with respect to time yields

$$\dot{\mathbf{e}} = \dot{\boldsymbol{\omega}} - \dot{\boldsymbol{\omega}}_d \tag{19}$$

It is desirable for the error dynamics to decay exponential over time, i.e. $\mathbf{e} \propto e^{-k_p t}$, and therefore the error rate equation is desired to have the following form:

$$\dot{\mathbf{e}} = -k_p \mathbf{e} \tag{20}$$

Then using Euler's equation and assuming disturbance torques are negotiable, Eq. 19 can be written as

$$\dot{\mathbf{e}} = J_{\text{SO}}^{-1} \left(\mathbf{T}_{\text{control}} - [\boldsymbol{\omega}\times] \, J_{\text{SO}}\boldsymbol{\omega} \right) - \dot{\boldsymbol{\omega}}_d \tag{21}$$

where $\mathbf{T}_{\text{control}}$ is the torque provide by the attitude actuator. Then for an exponentially decaying tracking error the desired torque expression becomes

$$\mathbf{T}_{\text{control}} = -J_{\text{SO}}^{-1} \left(\tau - [\boldsymbol{\omega}\times] \, J_{\text{SO}}\boldsymbol{\omega} \right) + \dot{\boldsymbol{\omega}}_d - k_p \mathbf{e} \tag{22}$$

This expression is used to calculate the torque required to maintain the desired pointing profile. The desired angular velocity profile, $\dot{\boldsymbol{\omega}}_d, \boldsymbol{\omega}_d$, is determined from the desired pointing directions used for the particular control followed by the SO. This work uses Sun pointing, Nadir pointing, and spin stabilized control profiles, each with a different desired angular velocity profile, $\dot{\boldsymbol{\omega}}_d, \boldsymbol{\omega}_d$.

Training Set on Real Data

This work investigates using real light curve observations taken from the Multichannel Monitoring Telescope (MMT) [30]. This data source is publicly available through astroguard.ru. For this work, the training data was developed using segments of 500 measurement samples taken from the MMT dataset for objects with TLE information. Their TLE numbers and TLE names label the objects in the MMT dataset, and from the TLE names, class information can be extracted. Three classes are used in this work, and these classes are Debris, Rocket Bodies, and Satellite. Figure 3 shows some representative examples of the MMT data used for training. From this figure, a clear difference can be seen from Debris, Rocket Bodies, and Satellite classes.

Convolutional Neural Network Classification

Deep Learning Methods: Convolutional Neural Networks

Over the past few years, there has been an explosion of machine learning algorithms [16, 17, 20, 31–33]. Such algorithms can learn from data to accomplish specific tasks (e.g., image recognition, object identification, natural language processing, etc.). Among the various available techniques, deep learning, comprising of methods and techniques to design and train multi-layer neural networks, has been playing a dominant role. In contrast to shallow networks, deep networks refer to a class of neural networks with more than one hidden layer. Among all possible systems, one of the most powerful deep architectures is called Convolutional Neural Networks (CNNs) [18]. CNNs, which are nominally applied to input images, have been used to classify an image or determining the particular content of an image by transforming the original image through a set of layers with specialized architectures to a class score. Indeed, the architecture of a CNN is designed to take advantage of the 2D or 3D $\left[\text{width, height, depth}\right]$ structure of an input image which is processed as pixels values. The basic CNN structure uses many types of layers which are 1) Convolutional

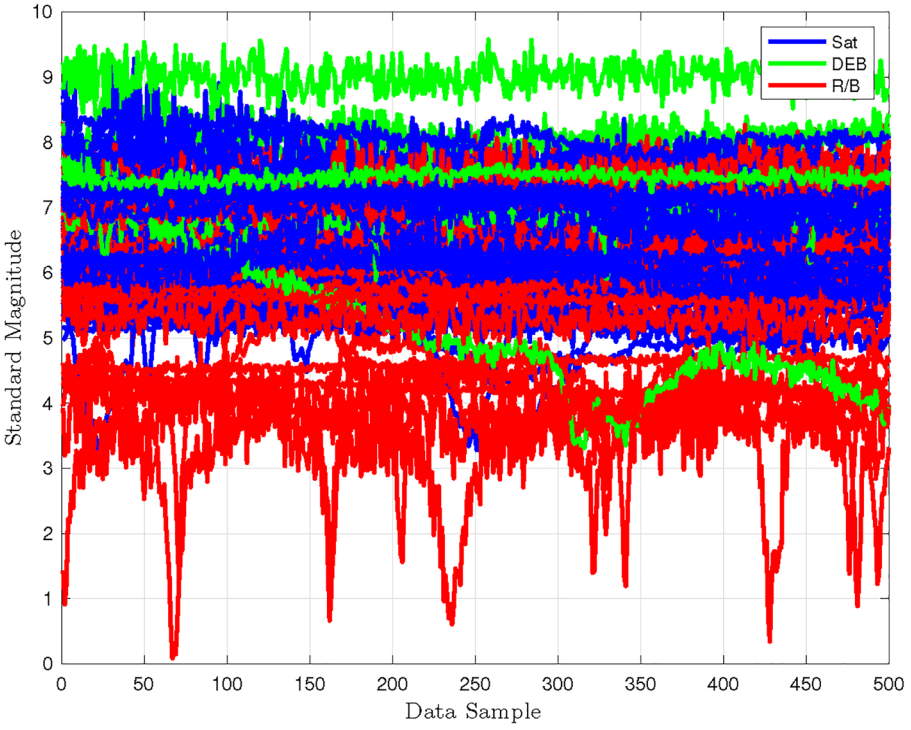


Fig. 3 This figure shows the labeled real training data from the MMT database, where each label is given a unique color

Layer, 2) Pooling Layer, 3) Fully Connected Layer, and 4) Output Layer. Reference 31 provides a comprehensive review of CNN based methods.

The core layer, i.e., the Convolutional layer, extract features from the input volume by applying filters on the image. It is the most demanding layer in terms of computations of a CNN, and the layer′s parameters consist of a set of learnable filters. Each filter is a matrix spatially smaller than the image, which is scanned along width and height (for the 2D input case). Importantly, filters (or kernels) are the weights of this layer. In fact, as the filter is sliding on the input image, it multiplies its values with the original pixel values of the image, and these multiplications are all summed up, giving only one number as output. Repeating this procedure for all the regions on which the filter is applied, the input volume is reduced and transformed and then passed to the *Max-pooling layers*. Mathematically, the convolutional layer can be described as follows:

$$(X * Y)(i, j) = \sum_{n=0}^{N} \sum_{m=0}^{M} X_{m,n} * W^{cov}_{i-m, j-n} \tag{23}$$

where $*$ is the convolution operation, W^{cov} is the convolution kernel corresponding to the randomly initialized weights, and X is the image with indices (m, n). CNNs typically employs the nonlinear activation function called $ReLU$ function (i.e., Rectified Linear Unit) described as $f(x) = \max(0, x)$. Spatial pooling layers group local features from spatially adjacent pixels to improve robustness. A set of convolutional layers are generally stacked below a fully connected layer that feeds a soft-max layer [32], which outputs the probability of the image belonging to one of the classes. CNNs are easier to train due to inherent parameter sharing in the convolutional layer. For a classification task, the cross-entropy function [32] is generally employed as the cost to minimize. CNNs are trained in batch mode via Stochastic Gradient Descent (SDG) [31] with variable size of mini-batches. The dropout technique improves generalization and avoids overfitting [16].

CNN Design, Training and Classification Results: Simulated Data

This section outlines the equations of the CNN method used in this work. For a complete discussion of CNN architectures please refer to Ref. [34]. As previously discussed, the training data set consists of pairs of simulated light curve measurement vectors (or real measurements) and class vectors. The input light curve and output class vector are denoted by $\mathbf{x} \in \mathbb{R}^{1\times m}$ and $\mathbf{y} \in \mathbb{R}^{1\times n_c}$, respectively, where m and n_c denotes the number of light curve measurements and number of classes, respectively. A CNN is trained to map the measurement vector, $\mathbf{x}$, to classes, $\mathbf{y}$, using a set of training examples. The vector $\mathbf{y}$ is the class labels for each light curve input $\mathbf{x}$. In this work the classes used are rocket bodies, payloads, and debris. For example, $\mathbf{y}$ in this case is a 3×1 vector and the elements represent the probability that the input light curve belongs to the rocket bodies, payloads, and debris classes, respectively. Then $\mathbf{y} = [0.95, 0.05, 0]^T$ indicates that the input light curve belongs to the rocket bodies class with probability 0.95, to the payload class with probability 0.05, and debris with probability 0. Since the labeled training set provides supervised examples, it has labels that indicate probability 1 for the class of the input light curve. However,

the neural network may have uncertainty in the class output and therefore does not typically have output labels of probability 1.

The CNN designed for this work to learn the functional relationship between measurements and SO classes consists of 1D convolutional layers with rectified linear unit (ReLU) activation, dropout, max-pooling, and two fully connected layer with ReLU activation (Fig. 4). The output layer uses softmax function to map to classification states. Each convolutional layer has the following form [16]:

$$\mathbf{h}^{cov}(\mathbf{x}) = \mathbf{f}\left(\mathbf{x} * W^{cov} + \mathbf{b}^{cov}\right) \tag{24}$$

Where $*$ denotes the convolution operator shown in Eq. 23, W^{cov} denotes the convolution kernel, and $\mathbf{f}$ is the activation function for each layer that adds nonlinearity to the feature vector. This work uses ReLU for convolutional layer activation. The ReLU function is given by $\mathbf{f}(\mathbf{x}) = \max(\mathbf{0}, \mathbf{x})$, where it is zero for negative input values and linear for positive input values. The convolution of $\mathbf{x}$ with W^{cov} defines the output feature map $\mathbf{h}^{cov}(\mathbf{x})$. The number of output maps is determined by the number of convolution filters for each convolutional layer. Each convolution layer has a collection of kernels of given size that are learned directly from the data. For the light curve problem the convolutions are of one dimensional time-series data. Then for input vectors having size $(1, s_x)$ the output vector is given by $(1, s_y)$ and the output vector size can then be calculated from the size of the kernel, s_k, and is given by $s_y = s_x - s_k + 1$. After the convolution is applied, the output vector is reduced in size, but zero padding is used in this work to keep the size of the feature vectors constant through each convolutional layer [34].

Then a CNN applies a series of these kernels W^{cov} in a layered fashion where each layer has a different size kernel that learns features on a given scale. To simplify the information in the output of each convolutional layer, max-pooling is used. In this work max-pooling with 1×4 kernel between each convolution layer is used. The max-pooling operation convolves the 1×4 kernel over the output of each convolutional layer returning the max of the outputs over the 1×4 region. Finally, at the final layer a nonlinear function is applied to the output using a traditional neural network. This final layer uses a fully connected neural network layer and is given by

$$\mathbf{h}^{fc}(\mathbf{x}) = \mathbf{f}\left(W^{fc}\mathbf{x} + \mathbf{b}^{fc}\right) \tag{25}$$

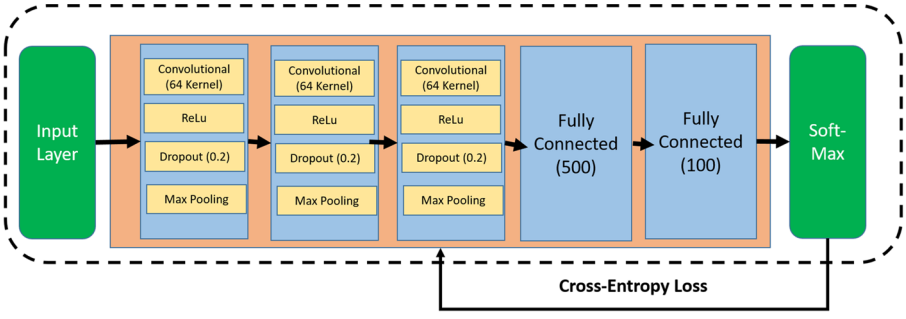


Fig. 4 Network architecture

After the fully connected layer a softmax function is used to provide outputs in the ranges (0, 1) that add up to 1. The softmax function is defined by

$$y_j\left(\mathbf{h}^{fc}(\mathbf{x})\right) = \frac{\exp(h_j^{fc})}{\sum_i \exp(h_i^{fc})} \tag{26}$$

The convolutional kernel and fully connected output layer parameters are arranged into the vector $\boldsymbol{\theta}$. The cost function used for this work is the cross-entropy loss. This loss function minimizes the cross-entropy loss between training outputs and the CNN outputs, and is given by [34]:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{N}\sum_{i=1}^{N} H(\mathbf{y}, \tilde{\mathbf{y}}) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{y}\log\tilde{\mathbf{y}} + (1-\mathbf{y})\log(1-\tilde{\mathbf{y}})\right] \end{aligned} \tag{27}$$

where $\tilde{\mathbf{y}}$ are the training examples and $\mathbf{y}$ are the outputs from the CNN. Then the CNN classification approach is trained by stochastic gradient descent by minimizing the cross-entropy loss from the outputs compared to the labeled data. Figure 4 shows the full architecture of the network used for this work. LeCun [32] showed that stochastic online learning is superior against the full batch mode as it is faster and results in more accurate solutions. The weights for the output layer and the convolutional layer are updated using the following relationship

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \eta\frac{\partial L}{\partial \boldsymbol{\theta}} \tag{28}$$

where t denotes the iteration step, and η is the learning rate. The $\frac{\partial L}{\partial \boldsymbol{\theta}}$ is the gradient of the loss with respect to the overall network parameters. This update is calculated for small batches over the entire training sets. Using the small batches allows for small updates to the gradient while reducing the noise in the gradient of individual training samples. This method of updating the parameters is referred to as stochastic gradient descent, and the gradient is calculated with error backpropagation.

t-SNE Technique for Data Dimensionality Reduction

The *t*riplet-distributed Stochastic Neighbor Embedding (*t*-SNE) is a probabilistic technique particularly suitable for visualization of high-dimensional data [35, 36]. The overall algorithm is designed to minimize the divergence between two distributions, i.e. between a) a distribution that measures the similarities between two input points pairwise, and b) a distribution that measures similarities of the corresponding points in the embedding as mapped pairwise on the low-dimensional subspace. The overall idea is to embed points living in a high-dimensional space into a low-dimensional space (generally 2-D or 3-D) in such a way that the mapping preserves similarities (distance) between points. Consequently, points that are nearby in the high-dimensional space are expected to be close in the low-dimensional embedding.

As such, one can visualize points in low-dimensional spaces to find natural clustering that occur in the high-dimensional spaces. The input data to the algorithm is the matrix $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \in \mathbb{R}^{D\times N}$ which contains N vectors $x_i \in \mathbb{R}^D$. After some preprocessing (e.g. subtraction of the mean and division by the standard deviation) the algorithm computes the distance $d(x_j, x_k)$ between each pair of points comprising the matrix X. The distance is then used to compute the *Perplexity* [35]. The latter is generally defined in terms of a model Gaussian distribution. Indeed, one defines the conditional probability of k given j as follows:

$$p(k|i) = \frac{\exp\left(-d(x_j,x_k)^2/(2\sigma_j^2)\right)}{\sum_{k\neq j}\exp\left(-d(x_j,x_k)^2/(2\sigma_j^2)\right)} \tag{29}$$

and

$$p(j|j) = 0 \tag{30}$$

Define now the joint probability p_{jk} as follows:

$$p_{ij} = \frac{p(j|k)+p(k|j)}{2N} \tag{31}$$

and the Shannon Entropy of P_j as follows:

$$H(P_j) = -\sum_k p(k|j)\log_2(p(k|j)) \tag{32}$$

Where the term P_j represents the conditional probability distribution over all data points x_i. The *Perplexity* $\text{Perp}(P_j)$, which measures the number of neighbors of point j, is defined as follows:

$$\text{Perp}(P_j) = 2^{H(P_j)} \tag{33}$$

Generally, the embedding of points contained in the matrix X into a low-dimensional space. The t-SNE algorithm executes an optimization procedure, i.e. it attempts to minimize the *Kullback-Leibler* (KL) divergence between the model Gaussian distribution of the points in X and a student t distribution of point Z in the selected low-dimensional space. Given the points $Z = \{z_1, z_2, ..., z_N\}$ in the selected low-dimensional embedding, the probability model of the distribution of the distances between two points can be described as follows:

$$q_{jk} = \frac{(1+\|z_j-z_k\|^2)^{-1}}{\sum_m\sum_{n\neq m}(1+\|z_m-z_n\|^2)^{-1}} \tag{34}$$

The KL divergence between th two distributions is therefore:

$$\mathcal{D}_{KL}(P\|Q) = \sum_{j\neq k} p_{jk}\log\frac{p_{jk}}{q_{jk}} \tag{35}$$

The minimization of the KL divergent is commonly executed by implementing a gradient descent approach.

Numerical Results

The CNN is trained over 8000 randomly generated scenarios comprising nine (9) possible classes of SOs. During the training, the CNN is validated against 5000

data scenarios not used in the training set. For all training scenarios, an SO is in near geosynchronous orbit with orbital elements given by $a = 42,364.17$ km, $e = 2.429 \times 10^{-4}$, $i = 30$ deg, $\omega = \Omega = 0.0$ deg and $M_0 = 91.065$ deg. The simulation epoch is 15-March-2010 at 04:00:00 GST. The initial quaternion and angular rate of the SO are given by

$$\mathbf{q}_I^B \equiv [0.7041 \;\; 0.0199 \;\; 0.0896 \;\; 0.7041]^T$$

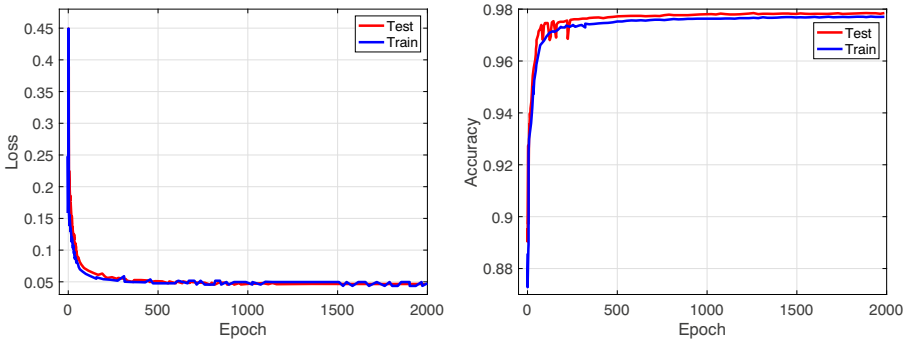
and

$$\boldsymbol{\omega}_{B/I}^B = [206.26 \;\; 103.13 \;\; 540.41]^T \text{ deg/hr}.$$

Brightness magnitude are simulated using a ground station located at 20.71° North, 156.26° West longitude and 3,058.6 m altitude. Measurements constructed using instantaneous geometry are corrupted by zero-mean Gaussian white noise with standard deviations of 0.1 for the brightness magnitude [37]. Observations are available every 5 seconds for about 1/4 hour (Fig. 2). All training samples are generated using the same orbit during the sample simulation time interval. Each sample has a different shape model, rotational initial condition, and a control profile.

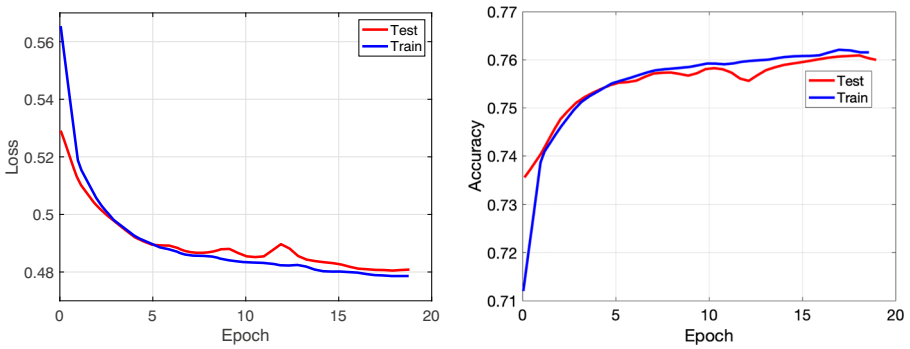
The Keras (Python) library with Tensorflow [38] as the backend is employed to design and train the network. The proposed CNN architecture is shown in Fig. 4. The CNN is comprised of a seven (7) layer structure, including the input layer, three convolutional layers, two fully connected layers, and one softmax layer. The input layer is made up of a light curve input vector comprising of 182 data points. The convolutional layers have 64, 32, and 64 filters, respectively. Both max-pooling (1×2 pooling kernel) and dropout are applied after each convolutional layer. For this work, the dropout regularization rates are set to be 0.2 for the convolutional layers and 0.5 for fully connected layers. Training occurs for 2000 epochs using stochastic gradient descent and a mini-batch of 128 samples. Figure 5 shows the results of the training process and the relative performance of the network (Fig. 6).

Both max-pooling (1×2 pooling kernel) and dropout are applied after each convolutional layer. The dropout rates used for this work are 0.7 and 0.5 for the convolutional and fully connected layers, respectively. Then the training data consists of simulated light curve measurements as inputs and class states as outputs. For



(a) CNN Loss (b) CNN Accuracy

Fig. 5 CNN classifications results simulated data case



(a) CNN Loss (b) CNN Accuracy

Fig. 6 CNN classifications results real-data case

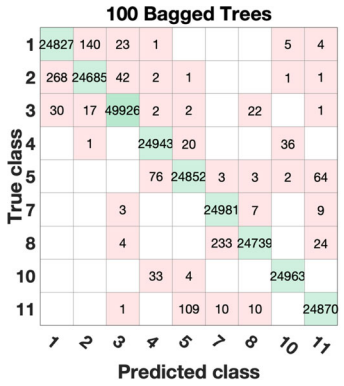
this study, we only consider shape classes with one control class, but other classes can be added in the same CNN or with independent CNNs for each class. The classes considered are rocket bodies, controlled payload, uncontrolled payload, and debris.

The 1D-CNN has been trained using a single GPU with 1500 processors. Figure 5 shows the results of the training process. The CNN has been trained on 8000 samples comprising the training set and tested on 2000 samples during the training process. Figure 5a shows the behavior of the cross-entropy loss as a function of the epoch. Figure 5b shows the model accuracy as a function of the epoch. Both test and training sets have a similar accuracy result. We report that the CNN exhibits an accuracy of 97.83% on the test set. Training time is reported to be 2000 sec.

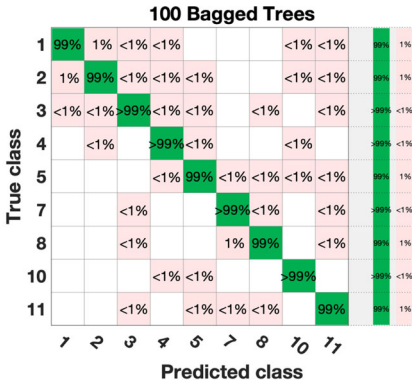
Importantly, we compared the CNN with two machine learning techniques that represent somewhat the state of the art of automatic classification techniques. The first method is called Bagged Trees, and it is an ensemble method. The second set of techniques are the well-known Support Vector Machines (SVM, [23]) adapted for the multi-class case

For the bagged decision trees, 100 weak learners (trees) have been selected. For each of the trees, we considered 140,000 as the maximum number of possible splits. The training set has been loaded in the MATLAB classification learner app. Data are pre-processed using Principal Component Analysis (PCA, [39]) for dimensionality reduction. To explain 98% of the variance, ten (10) principal components have been kept. A 5-fold validation approach has been implemented to protect against overfitting. The bagged model was trained in parallel fashion over an 8-core intel i7 operating @2.9Hz. The model achieved an accuracy of 99.5% over the training set. The training time was recorded to be 1858 sec. Figure 7a shows the resulting confusion matrix reported in terms of correct classification for the 250, 000 training cases. Figure 7b shows, per each individual class, the percentage of true positives rate and false-negative rate. Additionally, Fig. 7c shows the percentage of positive predicted values and the false discovery rate.

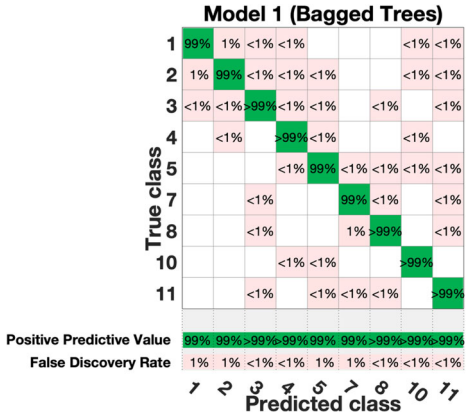
Similarly, an SVM with a cubic kernel has been selected for training. PCA and 5-fold validation has been applied, and the algorithm was run on the same machine.



(a) Shows the resulting confusion matrix reported in terms of correct classification for the 250,000 training cases



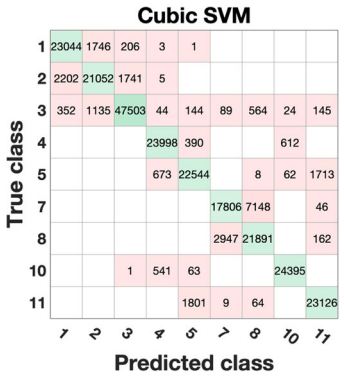
(b) Shows, per each individual class, the percentage of true positives rate and false-negative



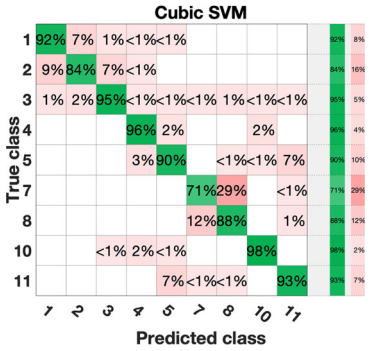
(c) Shows the percentage of positive predicted values and the false discovery rate

Fig. 7 Bag of tree simulation cases

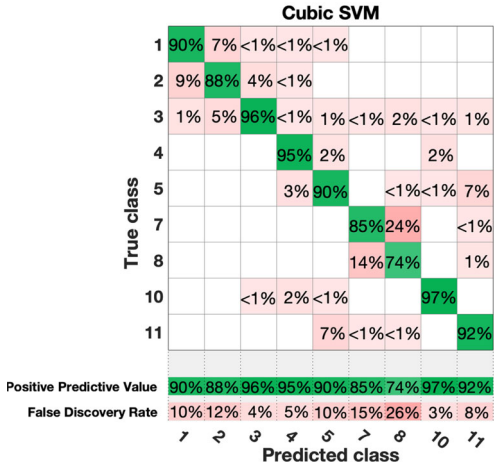
The SVM model achieved an accuracy of 90.1% over the training set. The training time was recorded to be 25774 sec. Figure 8a shows the resulting confusion matrix reported in terms of correct classification for the 250, 000 training cases. Figure 8b shows, per each individual class, the percentage of true positives rate and false-negative rate. Additionally, Fig. 8c shows the percentage of positive predicted values and the false discovery rate. Overall, for the simulated data, the accuracy between deep and non-deep methods is generally comparable.



(a) Shows the resulting confusion matrix reported in terms of correct classification for the 250,000 training cases



(b) Shows, per each individual class, the percentage of true positives rate and false negative rate



(c) Shows the percentage of positive predicted values and the false discovery rate

Fig. 8 Support vector machine simulation cases

CNN Design, Training and Classification Results: Real Data

The same 1D-CNN architecture described above (see Fig. 4) has been employed to train the deep network on the real data. In this case, the training set comprised 10, 000 samples and is subdivided into three possible classes, i.e., rocket bodies, debris, and other. In the class other, it is included anything that is not either a rocket body or debris. A set of 2261 samples is employed as a test set during the training. The input

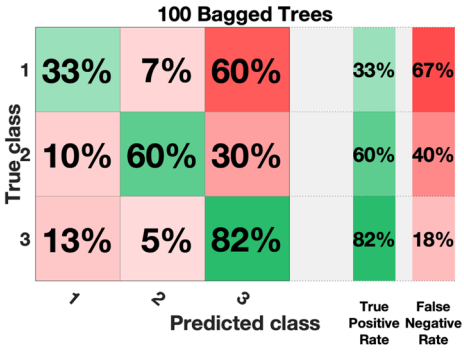
light curve vector comprises 500 points. The number of epochs is set to be 2000. Figure 6 shows the results of the training process. Figure 6a shows the behavior of the cross-entropy loss as a function of the epoch. Figure 6b shows the model accuracy as a function of the epoch. The real data set is more comprehensive as it accounts for classes of objects with different observing conditions and different orbits (i.e., GEO, LEO, and MEO). Thus, it is expected that the separation boundaries between classes are highly non-linear. Here, we show that accuracy over the training set and test set is markedly different. At the end of the training, accuracy on the sequence of mini-batches is as large as 87.5%. Conversely, the final accuracy on the test set is 75.4%. Training time is about 4000 sec and a single GPU with 1500 processors.

For comparison and similarly to the simulated light curve case, bagged trees (ensemble of 100 weak learners) and SVM with the cubic kernel are developed and trained on the same training set comprising real data. PCA has been considered to retain the principal components that can explain 98% of the covariance in the data, resulting in 21 components out of 500. For both cases, we considered a 5-fold method as protection against overfitting. Performance for both cases is reported to be rather poor. Indeed, bagged trees achieve accuracy on the training set of 62.6% (training time: 143 sec), whereas SVM achieves an accuracy of 44.4% (training time: 1894 sec). Figure 9a shows the resulting confusion matrix reported in terms of correct classification for the 147,737 training cases. Figure 9b shows, per each individual class, the percentage of true positives rate and false-negative rate. Additionally, Fig. 9c shows the percentage of positive predicted values and the false discovery rate. Similarly, Fig. 10a shows the resulting confusion matrix reported in terms of correct classification for the 147, 737 training cases. Figure 10b shows, per each individual class, the percentage of true positives rate and false-negative rate. Additionally, Fig. 10c shows the percentage of positive predicted values and the false discovery rate. 1D-CNN performs better due to the superior ability to automatically extract a hierarchical set of features comprising the light curve signal.

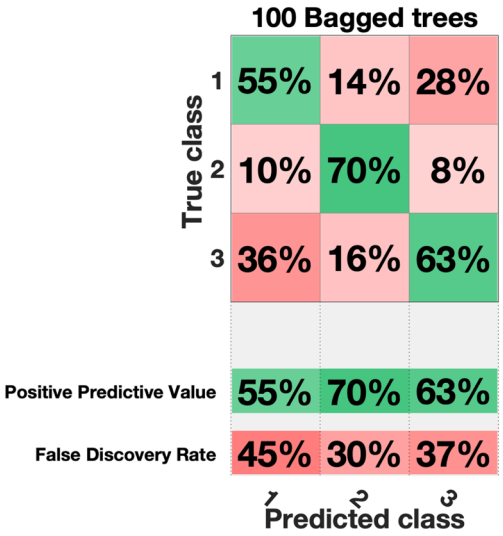
Training Set Visualization and Clustering

Understanding the training data structure may provide insight into the ability of the deep classifier to recognize the right SO after training. However, the visualization of high dimensional data sets is generally a difficult problem. Indeed, the measured light-curves employed in the training set are represented as one-dimensional vectors comprising 500 components. Likewise, the simulated data include 182 components. A conventional approach to data reduction and visualization considers processing the data using PCA, where the most significant two or three components can be visualized in two-dimensional and three-dimensional spaces. However, as a linear technique, PCA captures only the linear structures in the data and generally linearly maps high dimensional spaces into low dimensional subspaces. Recently, new probabilistic approaches that preserve the local distance between data while mapping the high-dimensional data into a low-dimensional subspace have been introduced. One of the most popular approaches to such non-linear data dimensionality reduction is the t-Distributed Stochastic Neighbor Embedding (t-SNE, [35]). We have considered

(a) Bag of Tree Real Data 1



(b) Bag of Tree Real Data 2



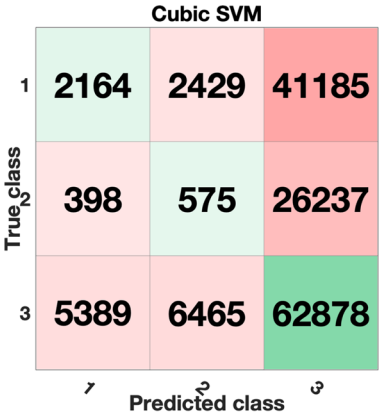
(c) Bag of Tree RealData 3

Fig. 9 Bag of tree measured data cases

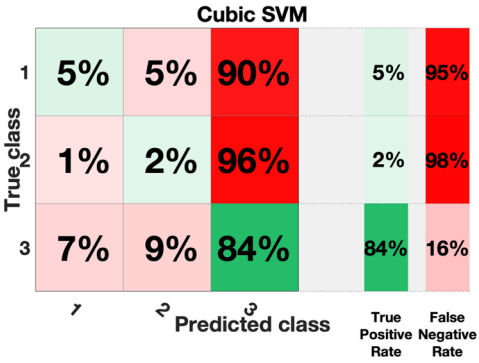
such a technique for the visualization of the training data in two and three-dimensions to understanding the structure and clustering of the different RSO classes employed during the learning processes by our deep networks.

t-SNE Visualization of the Measured and Simulated Training Data

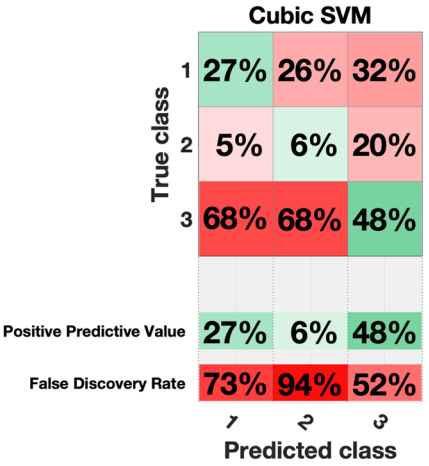
We considered 2-D and 3-D visualization of the simulated and measured light-curves employed in the training of the deep networks. First, the 2-D embedding of sampled



(a) SVM Real Data 1



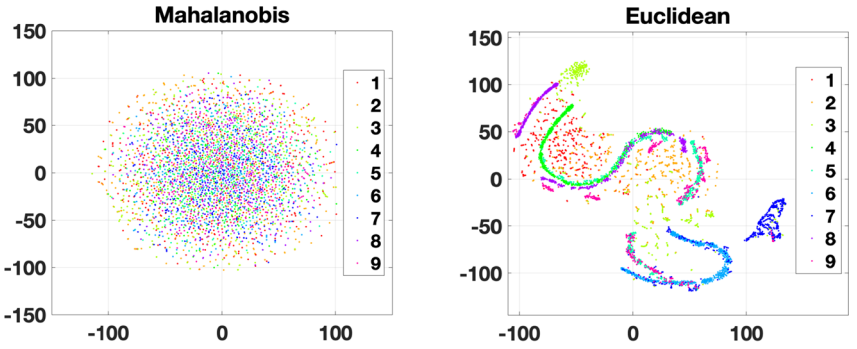
(b) SVM Real Data 2



(c) SVM Real Data 3

Fig. 10 Support vector machine measured data cases

simulated training points is considered for visualization. In this case, the training set comprises 250,000 simulated light-curves distributed among 9 possible classes. A vector of dimension $D = 182$ represents each light curve. We sampled 3600 vectors (i.e., 400 per class) and processed the sample via the t-SNE algorithm for four selected distances, i.e., Mahalanobis, cosine, Chebyshev, and Euclidean. The resulting 2-D embedding is visualized in Fig. 11a, b, c and d. Structure of the data and clustering is readily apparent in all cases but the one employing the Mahalanobis distance. All classes seem to be fairly separated in the 2-D embedding, which reflects the clustering of the 182-dimensional space of the original light-curves. The separation between the different classes may be responsible for the high performances



(a) The Maha Two-Dimensional Embedding

(b) The Euclidean Two-Dimensional Embedding

(c) The Euclidean Two-Dimensional Embedding

(d) The Cosine Two-Dimensional Embedding

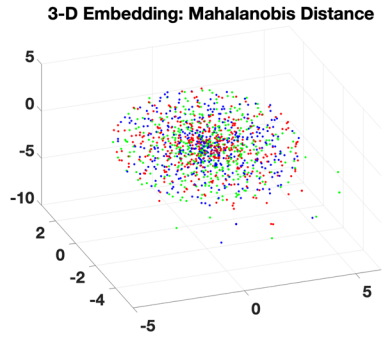
Fig. 11 The two-dimensional embedding of simulated case

achieved with a non-deep machine learning technique (e.g., bagged decision trees), which is able to find the decision boundaries between the different classes easily. Although the deep CNN achieves similar performances, the deep architecture does not yield any significant advantage for the case generated by the simulated data. The situation is completely different for the training set generated from measured light curves. Here, the 3-D embedding of sampled measured training points is considered for visualization. In this case, the training set comprises $150,000$ measured light curves distributed among 3 possible classes. A vector of dimension $D = 500$ represents each light curve. We sampled 1200 vectors (i.e., 400 per class) and processed the sample via the t-SNE algorithm for four selected distances, i.e., Mahalanobis, Cosine, Chebyshev, and Euclidean. The resulting 3-D embedding is visualized in Fig. 12a, b, c and d. As in the previous case, the structure of the data and the corresponding clustering in 3-D is readily apparent in all cases but the one employing the Mahalanobis distance. Here the clustering of data in the 3D embedding is much more complex, and decision boundaries are not well defined as in the previous case. As

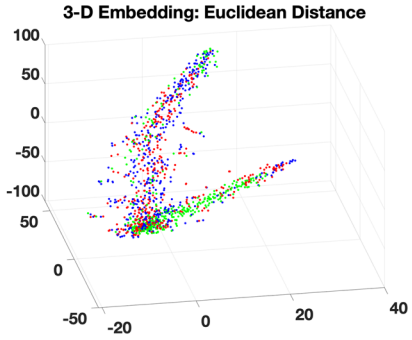
seen in the previous sections, the non-deep machine learning techniques (i.e., bagged decision trees and SVM) dramatically underperform the deep CNN architecture. Due to many-layer configuration, the CNN can learn complex decision boundaries and discriminate between the three classes in a much more efficient fashion.

Limitation of Deep Learning and Future work

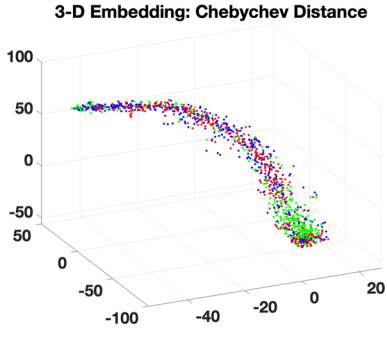
The current work shows promise of using the CNN approach for classifying light curve measurements, but the approach does have some limitations that need to be investigated for future studies. The main limitation of the current work is ensuring that training data for the class is sampled over a wide enough distribution. Since the light-curves are dependent on orientation, lighting conditions, surface materials, and shape, among other parameters, methods for ensuring that the training data is sampled over a wide enough distribution are needed. Future work may investigate how real-data examples could aid in defining such a distribution. Additionally, the choice of the BRDF model affects the performance of the simulation-based CNN approach.



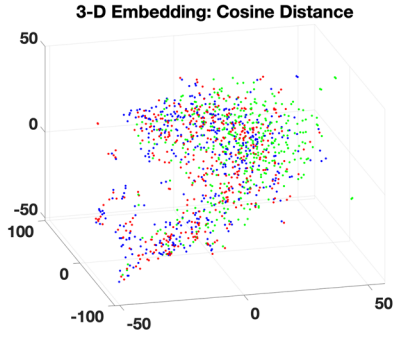
(a) The Mahalanobis Three-Dimensional Embedding



(b) The Euclidean Three-Dimensional Embedding



(c) The Chebychev Three-Dimensional Embedding



(d) The Cosine Three-Dimensional Embedding

Fig. 12 The three-dimensional embedding of the real-data case

This limitation can be overcome by using other BRDF models [14] to generate the training set in addition to the AS BRDF model. Finally, the hyper-parameters (such as number of layers, number of neurons per layer, and type of activation function used) have a significant effect on the performance of the CNN-based approach. This work did not explore optimizing these hyper-parameters in any systematic way.

A critical future work item is understanding how the results on the simulated data case can be transferred to the real-data case. The 1D-CNNs have been designed and trained separately on different data sets. The first dataset is comprised of simulated data generated using a physically-based reflectance model. The second dataset has been extracted from real light curve data. Although a set of 10,000 light-curves have been simulated, one can potentially generate much larger datasets comprising millions of light curves with automatic labeling. Indeed, the physical model can be employed to produce a large number of classes of SOs, each with light-curves simulated under different conditions (e.g., observing conditions, orbital regimes, material reflectance models, etc.). The physical model enables controlled data generation and ensures sufficient training points necessary to model the complex relationship between light-curves and classes. However, physically-based models are affected by *modeling error*, which may be a limitation, especially when applied to classify SO based on observed light-curves. It is expected that although a CNN performs well on simulated light-curves, it may not perform well when, after training, it is applied to predict the SO class. Conversely, collected data are generally scarce. Since CNNs are comprised of millions of trainable parameters, they tend to overfit the data. Importantly, since deep architectures autonomously learn the relevant features in a data-driven hierarchical fashion, a CNN trained on the physically-based model may have learned the basic features. This approach is conducive to the Deep Transfer Learning [40], where deep networks are pre-trained on a significant source of data and then applied to a new but limited target set [40]. In transfer learning, one first trains a base network on a base dataset and task; subsequently, one transfers the learned features on a smaller target set trained on a much smaller dataset. Generally, this process works well if the learned features are general, i.e., capture the fundamental behavior of the system. It is conjectured that if one employs the simulated light-curves as base training set to train a physically-based CNN, the learned features may capture the basic physics and may be general enough to be successfully transferred to a target network trained on a much smaller dataset, yet yielding superior accuracy. In general, training the target network may be much faster and efficient. This conjecture is currently investigated and may be reported in future works.

Conclusion

In this paper, a data-driven classification approach based on the Convolutional Neural Network (CNN) scheme is used for Space Object (SO) classification using light curve data. The classification approach determines the class from light curve observations of a SO. These classes are rocket bodies, payloads, and debris. A set of 1D-CNNs capable of ingesting light-curves is separately trained on both simulated data (generated by a physics-based model) and real observed light-curves and the performances

are reported. It is shown that CNNs are highly accurate classifiers whenever trained on simulated data, yielding 98% accuracy for the selected test set. However, whenever trained on simulated light-curves, the CNN tends to lose the advantage over more conventional state-of-the-art machine learning methods (e.g., bagged trees, SVM). Nevertheless, the CNN's significantly outperforms the other techniques whenever real data are considered. Indeed, on real light curve test sets, CNNs achieve 75% accuracy, whereas bagged trees report an accuracy of 62% and SVM an accuracy of 44%. These results are promising for the application of CNN-based methods to Space Situational Awareness (SSA) applications.

Acknowledgment The first author wishes to acknowledge support of this work by the Air Force's Office of Scientific Research under Contract Number FA9550-18-1-0115.

Compliance with Ethical Standards

Conflict of interests On behalf of all authors, the corresponding author states that there is no conflict of interest.

Appendix A: Angular Velocity Determination Method

This appendix provides a summary of the angular velocity determination method discussed in Ref. [28]. This work uses three possible control profiles, Sun pointing, Nadir pointing, and spin-stabilized. For each of these profiles the SO is control led to point to the Sun direction, the Nadir, or in a spin-stabilized configuration. For the spin-stabilized configuration, the desired angular velocity is chosen to be perpendicular to the orbital plane. When pointing in Sun direction or the Nadir direction we must compute the desired angular velocity to track these directions. If we assumed that $\tilde{\mathbf{b}}_{j_k}$ and $\tilde{\mathbf{b}}_{j_{k+1}}$ represent the j^{th} pointing directions at time step k and $k+1$, then the goal is to estimate the angular velocity, $\boldsymbol{\omega}_k$, from these directions. Both the Sun pointing and Nadir directions pointing are determined from the SO trajectories. The $\boldsymbol{\omega}_k$ is then estimated using a finite difference method which is outline below [28]. Ċonsider the following unit-vector measurement model at time t_k:

$$\tilde{\mathbf{b}}_{j_k} = A_k \, \mathbf{r}_j + \mathbf{v}_{j_k} \tag{36}$$

where $\tilde{\mathbf{b}}_{j_k}$ is the j^{th} pointing vector in the inertia frame and is $\mathbf{r}_j$ the same pointing vector in the body frame. The attitude matrix mapping from inertial to the body frame is denoted by A_k. The goal is to determine the rate of change of this attitude matrix or the angular velocity. Taking the difference between successive measurements of Eq. 36 gives

$$\tilde{\mathbf{b}}_{j_{k+1}} - \tilde{\mathbf{b}}_{j_k} = [A_{k+1} - A_k] \, \mathbf{r}_j + \mathbf{v}_{j_{k+1}} - \mathbf{v}_{j_k} \tag{37}$$

We assume that the body angular velocity $\boldsymbol{\omega}$ is constant between t_k and t_{k+1}, and ignore terms higher than first order in $\boldsymbol{\omega}\,\Delta t$. With these assumptions the following first-order approximation can be used [26]:

$$A_{k+1} \approx \left[I_{3\times 3} - \Delta t\,[\boldsymbol{\omega}_k\times]\right]\,A_k \tag{38}$$

In this case $\boldsymbol{\omega}_k$ is the *average* velocity, but this becomes less of a problem as the sampling interval decreases. Substituting Eq. 38 into Eq. 37 gives

$$\tilde{\mathbf{b}}_{j_{k+1}} - \tilde{\mathbf{b}}_{j_k} = -\Delta t\,[\boldsymbol{\omega}_k\times]\,A_k\,\mathbf{r}_j + \mathbf{v}_{j_{k+1}} - \mathbf{v}_{j_k} \tag{39}$$

Our goal is to determine an angular velocity just using $\tilde{\mathbf{b}}_{j_k}$ and $\tilde{\mathbf{b}}_{j_{k+1}}$. This is accomplished by solving Eq. 36 in terms of $A_k\,\mathbf{r}_i$ and substituting the resultant into Eq. 39, which yields

$$\frac{1}{\Delta t}[\tilde{\mathbf{b}}_{j_{k+1}} - \tilde{\mathbf{b}}_{j_k}] = [\tilde{\mathbf{b}}_{j_k}\times]\,\boldsymbol{\omega}_k + \mathbf{w}_{j_k} \tag{40}$$

where $\mathbf{w}_{j_k}$ is the new effective measurement noise vector but for this work we assume this is zero. Note that Δt will have finite values, since discrete-time measurements are assumed. Equation 40 can now be cast into a linear least-squares form for all measurement vectors, which leads to

$$\begin{aligned}\hat{\boldsymbol{\omega}}_k \;=\; &\tfrac{1}{\Delta t}\left[\textstyle\sum_{j=1}^{n_k}[\tilde{\mathbf{b}}_{j_k}\times]^T R_{j_k}^{-1}[\tilde{\mathbf{b}}_{j_k}\times]\right]^{-1}\\ &\textstyle\sum_{j=1}^{n_k}[\tilde{\mathbf{b}}_{j_k}\times]^T R_{j_k}^{-1}(\tilde{\mathbf{b}}_{j_{k+1}} - \tilde{\mathbf{b}}_{j_k})\end{aligned} \tag{41}$$

where $\hat{\boldsymbol{\omega}}_k$ is the estimate of $\boldsymbol{\omega}_k$. For this work we assume $R_{j_k}^{-1} = I_{3\times 3}$.

References

1. Abbot, R.I., Wallace, T.P.: Decision support in space situational awareness. Lincoln Laboratory Journal **16**(2), 297 (2007)
2. Kelso, T.: "Satcat boxscore," Celestrak: SATCAT Boxscore. Accessed April 19, vol. 16 (2017)
3. Linares, R., Jah, M.K., Crassidis, J.L., Nebelecky, C.K.: Space Object Shape Characterization and Tracking Using Light Curve and Angles Data, J. Guid. Control. Dyn. **37**(1), 13 (2013)
4. Hall, D., Calef, B., Knox, K., Bolden, M., Kervin, P.: Separating Attitude and Shape Effects for Non-resolved Objects, In: The 2007 AMOS technical conference proceedings, pp. 464–475 (2007)
5. Jah, M., Madler, R.: Satellite Characterization: Angles and Light Curve Data Fusion for Spacecraft State and Parameter Estimation, In: Proceedings of the advanced Maui optical and space surveillance technologies conference, vol. 49 (Wailea, Maui, HI). Paper E49 (2007)
6. Holzinger, M.J., Alfriend, K.T., Wetterer, C.J., Luu, K.K., Sabol, C., Hamada, K.: Photometric attitude estimation for agile space objects with shape uncertainty. J. Guid. Control. Dyn. **37**(3), 921 (2014)
7. Linares, R., Shoemaker, M., Walker, A., Mehta, P.M., Palmer, D.M., Thompson, D.C., Koller, J., Crassidis, J.L.: Photometric Data from Non-Resolved Objects for Space Object Characterization and Improved Atmospheric Modeling. In: Advanced Maui optical and space surveillance technologies conference, vol. 1, vol. 1, p. 32 (2013)
8. Linares, R., Jah, M.K., Crassidis, J.L., Leve, F.A., Kelecy, T.: Astrometric and photometric data fusion for inactive space object feature estimation. In: Proceedings of 62nd international astronautical congress, international astronautical federation, vol. 3, pp. 2289–2305 (2011)
9. Gaylor, D., Anderson, J.: Use of Hierarchical Mixtures of Experts to Detect Resident Space Object Attitude. In: Advanced Maui optical and space surveillance technologies conference, vol. 1, p. 70 (2014)

10. Linares, R., Jah, M.K., Leve, F.A., Crassidis, J.L., Kelecy, T.: Astrometric and Photometric Data Fusion For Inactive Space Object Feature Estimation. In: Proceedings of the international astronautical federation (Cape Town, South Africa). Paper ID: 11340 (2011)
11. Linares, R., Jah, M.K., Crassidis, J.L.: Inactive Space Object Shape Estimation via Astrometric And Photometric Data Fusion. In: AAS/AIAA Space Flight Mechanics Meeting (Charleston, SC), AAS Paper 2012-117 (2012)
12. Linares, R., Jah, M.K., Crassidis, J.L.: Space Object Area-To- Mass Ratio Estimation Using Multiple Model Approaches. Adv. Astronaut. Sci. **144**, 55 (2012)
13. Hinks, J.C., Linares, R., Crassidis, J.L.: Attitude Observability from Light Curve Measurements. In: AIAA Guidance, navigation, and control (GNC) conference (AIAA, Boston, MA) (2013). https://doi.org/10.2514/6.2013-5005
14. Wetterer, C.J., Linares, R., Crassidis, J.L., Kelecy, T.M., Ziebart, M.K., Jah, M.K., Cefola, P.J.: Refining space object radiation pressure modeling with bidirectional re ectance distribution functions. J. Guid. Control. Dyn. **37**(1), 185 (2013)
15. Linares, R., Crassidis, J.L.: Resident Space Object Shape Inversion via Adaptive Hamiltonian Markov Chain Monte Carlo. In: AAS/AIAA Space Flight Mechanics Meeting (Napa, CA), AAS Paper 2016-514 (2016)
16. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
17. Lee, H., Pham, P., Largman, Y., Ng, A.Y.: Unsupervised feature learning for audio classification using convolutional deep belief networks. In: Advances in neural information processing systems, pp. 1096–1104 (2009)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
19. Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database. In: Advances in neural information processing systems, pp. 487–495 (2014)
20. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1725–1732 (2014)
21. Dietterich, T.G.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems. Springer, pp. 1–15 (2000)
22. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81 (1986)
23. Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J., Scholkopf, B.: Support vector machines. IEEE Intell Syst Appl **13**(4), 18 (1998)
24. Breiman, L.: Random forests. Mach Learn **45**(1), 5 (2001)
25. Ashikmin, M., Shirley, P.: An Anisotropic Phong Light Reflection Model. Tech. Rep. UUCS-00-014, University of Utah, Salt Lake City, UT (2000)
26. Shuster, M.D.: A Survey of Attitude Representations. J. Astronaut. Sci. **41**(4), 439 (1993)
27. Kaasalainen, M., Torppa, J.: Optimization methods for asteroid lightcurve inversion: I. shape determination. Icarus **153**(1), 24 (2001)
28. Linares, R., Crassidis, J.L., Jah, M.K.: Space object classification and characterization via multiple model adaptive estimation. In: 17th International conference on information fusion (FUSION) (IEEE), pp. 1–7 (2014)
29. Schaub, H., Junkins, J.L.: Analytical mechanics of space systems (American Institute of Aeronautics and Astronautics (2005)
30. Beskin, G., Karpov, S., Biryukov, A., Bondar, S., Ivanov, E., Katkova, E., Orekhova, N., Perkov, A., Sasyuk, V.: Wide-field optical monitoring with Mini-MegaTORTORA (MMT-9) multichannel high temporal resolution telescope. Astrophys. Bull. **72**(1), 81 (2017)
31. Rawat, W., Wang, Z.: Deep convolutional neural networks for image classification: A comprehensive review. Neural Comput. **29**(9), 2352 (2017)
32. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278 (1998)
33. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
34. Bengio, Y., Goodfellow, I., Courville, A.: Deep learning, vol 1 (Citeseer (2017)
35. Maaten, L.v.d., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579 (2008)

36. Van Der Maaten, L., Weinberger, K.: Stochastic triplet embedding. In: IEEE international workshop on machine learning for signal processing. IEEE, vol. 2012, pp. 1–6 (2012)
37. Hall, D.T., Africano, J.L., Lambert, J.V., Kervin, P.W.: Time- Resolved I-Band Photometry of Calibration Spheres and NaK Droplets. J. Spacecr. Rocket. **44**(4), 910 (2007)
38. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467 (2016)
39. Jolliffe, I.: Principal component analysis. In: International encyclopedia of statistical science. Springer, pp. 1094–1096 (2011)
40. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks?. In: Advances in neural information processing systems, pp. 3320–3328 (2014)