Galaxies Far, Far, Away

New Mexico

Supercomputing Challenge

Final Report

April 5, 2023

Team Number 50

St. Thomas Aquinas School

Team Members

Catherine Sedillo

Teacher

Eric Vigil

Project Mentor

James Sedillo

Contents

Executive Summary

Modern astronomical instruments are revealing an endless number of galaxies everywhere observations are made. This creates an overwhelming volume of information beyond the capability of human effort. With recent improvements in computing power and machine learning, the convolutional neural network shows promise as an aid to the astronomical surveyor especially for galaxy classification.

The capabilities of machine learning for galaxy classification are demonstrated here using Python, the Tensorflow 2 module, the Galaxy Zoo 2 dataset, and a small convolutional neural network model. A custom-method of image preprocessing is detailed and applied to the imagery to aid the convolutional neural network with achieving a high training accuracy.

Final training and validation of the convolutional neural network developed here resulted in a training accuracy of 98% and a validation accuracy near 80%. A simple graphical user interface (GUI) was made to allow the user to train, save, load, and interact with the CNN model and observe its class predictions in real-time.

Introduction

Modern astronomical telescopes are looking deeper into the universe. The images that they produce reveal the existence of countless stars and galaxies. The seemingly infinite quantity of these astronomical objects in these images has become too much for humans to identify and characterize. Machine learning can provide the ability to automatically classify these astronomical objects, allowing astronomers to focus on more abstract concepts.

This project uses a Convolutional Neural Network (CNN) architecture combined with a preprocessing technique to train a neural network to classify galaxies in the Galaxy Zoo 2 dataset. The goal is to achieve high-accuracy predictions of the galaxy classification.

Methodology

Neural networks are popular for recognizing and classifying objects in images. Common uses include video surveillance, self-driving vehicles, and cancer screening. The goal of this project is to apply this capability for the purpose of identifying galaxy types in telescope imagery using a neural network topology called the Convolutional Neural Network, or CNN.

Convolutional Neural Networks

Convolutional Neural Networks make up the first layers of a neural network intended for image recognition. This is because, like the animal visual cortex, CNNs provide a feature-extraction capability to the neural network (NN)[4, p. 518]. The output of a CNN layer is typically connected to a "max pooling" layer creating what is referred to as a "feature map." The layering of CNNs and max-pooling layers provides the broader neural network with the capability to recognize more complex image features. A set of fully-connected neural network layers make-up the final layers of the CNN with the last layer composed of a number neurons corresponding to the number of object classes the CNN needs to identify.

Image Dataset and Image Classes

Training a CNN must be done with the aid of a dataset that has an abundance of images which show the objects of interest. For this project, the dataset was downloaded from the Galaxy Zoo 2 dataset available in [1]. This dataset contains 243,437 images obtained by the Sloan Digital Sky Survey (SDSS) telescope. This dataset features a multitude of galaxy types in color, 424x424 resolution, JPEG images along with

two spreadsheets for referencing each image's classification. Image classes include the following examples [5, p. 2860]:

- Er = smooth galaxy, completely round.
- SBc2m = barred disk galaxy with a just noticeable bulge and two medium-wound spiral arm(s).
- Seb = edge-on disk galaxy with a boxy bulge.
- Sc(I) = disk galaxy with a just noticeable bulge, no spiral structure, and irregular morphology.
- A = star.

For this project, the classes were reduced to 3: elliptical galaxies, spiral/disk galaxies, and edge-on spiral/disk galaxies. Stars and galaxies with irregular features were not included in the training and validation datasets to improve training accuracy.

Software and Hardware

The software was developed using the Python programming language running within an Anaconda environment. Python modules used included the following:

- OpenCV: for loading images, grayscale-conversion, and saving images.
- Scikit-Image: for generating intensity profile lines in images.
- tkInter: for application GUI rendering.
- Numpy: for list and matrix math.
- Pandas: for reading *.csv files
- Matplotlib: for plotting/visualizing data.
- Tensorflow 2: for building, training, and testing the CNN.
- os: for image directory queries.
- Random: for image file name shuffling.

Hardware specifications of the PC were:

- Intel i9-9900KF processor
- 64-GB system RAM.
- nVidia RTX-2080 Super graphics card with 8GB VRAM.
- Windows 10 OS.

Data Preprocessing

Data preprocessing can be any method that helps remove unnecessary information and/or enhances relevant information within an image. According to [2, p.13], "In an image-understanding system, the preprocessing stage often performs functions such as the gray scale manipulation, edge detection, developing descriptions of objects or shapes in the image, image restoration, and geometric correction." The use of preprocessing allows the Neural Network to train on relevant information and comes highly recommended from the research literature.

As suggested by [2, p.13], the galaxy images were first converted to grayscale to reduce the image dimensionality from 3D-RGB (Red, Green, Blue) to 2D-grayscale. This resulted in 424x424-pixel images with intensity values ranging from 0 to 255. Since the images contain an abundance of extra objects such as stars and other galaxies, an approach was devised to remove these objects by interpreting the image as a topographical map. Topographical maps feature contour lines representing locations of equal altitude. For the galaxy images, contour lines represent pixels of equal intensity. Using the scikit-image module available to Python, the find_contours() function was employed to generate contours for pixels of intensity 60 (out of 255). Of the many contours generated by this function, the contour with the largest length was chosen as it was most likely to correlate with the largest object in the image, this being the galaxy of interest. A simple computation of the hypotenuse on each of this galaxy contour's set of coordinate pairs resulted in a list of radius values of the galaxy contour. The largest of these radii was then used as the radius for creating a circular image filter where pixels within the circle had value 1 and all pixels outside the filter had a values of 0. The image filter was made to have the same dimensions as the original image (424x424) with the circular filter overlapping the galaxy. Multiplication of the corresponding pixels of the image and the image filter generally removed most of the objects within the image while keeping the galaxy visible. Examples of this are shown in the figures 1 and 2. The Python application developed for this purpose (countour_prep.py) was used to preprocess a large batch of 8,443 image files for use by the CNN.

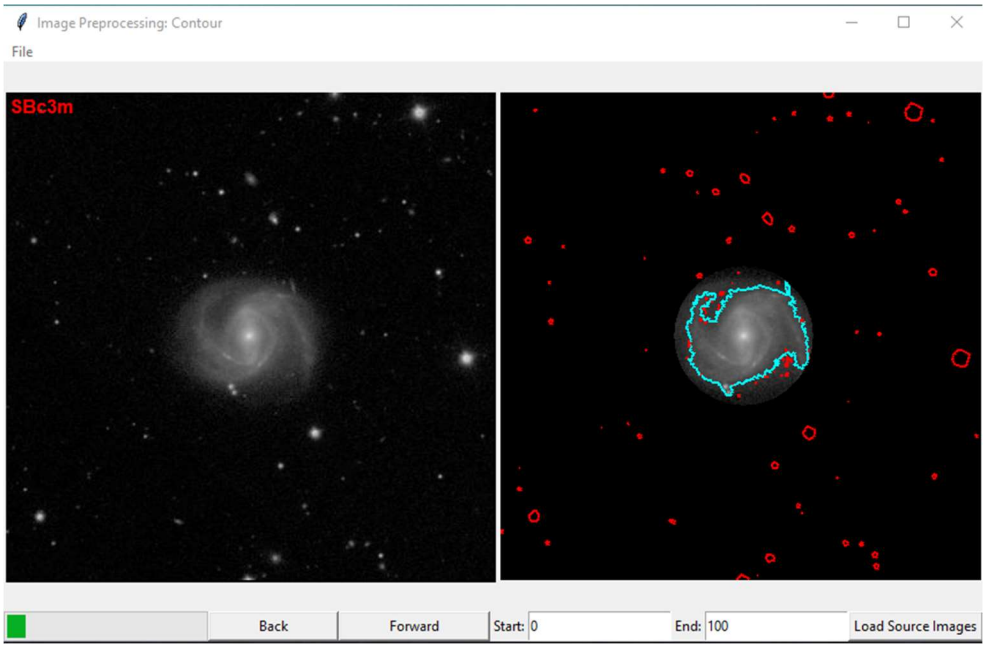

Figure 1 Preprocessing applied to a spiral galaxy image.

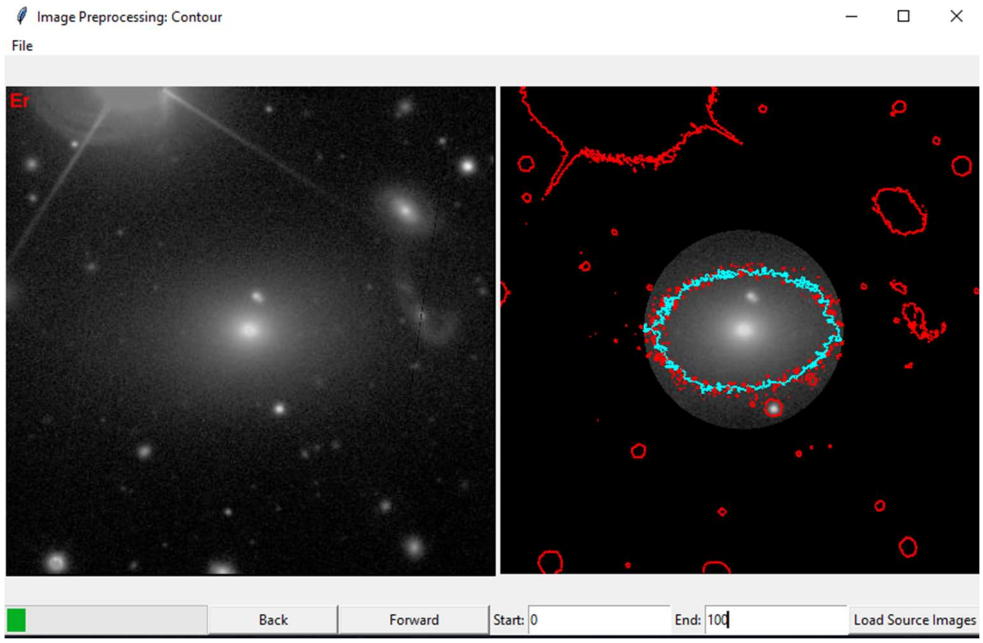

Figure 2 Preprocessing applied to an elliptical galaxy image.

Model and Verification Methods

Convolutional Neural Network Architecture

Various CNN architectures were tested in this study in order to obtain one with the highest validation accuracy. The final architecture tested took the form of figure 3.

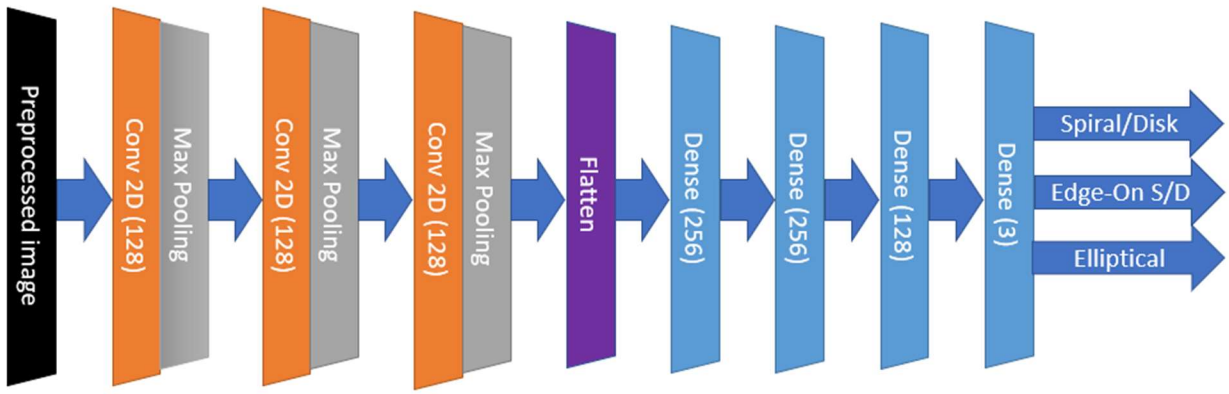

Figure 3 Final CNN architecture

As can be seen from the figure, the CNN takes a preprocessed image at its input. This image is then presented to a convolutional layer for the first feature-extraction phase. Since convolutional layers increase the dimensionality of the data, max-pooling layers are added behind the convolutional layers to reduce or "downsample" the data [3, p.385]. Two additional convolutional layers are added to provide the CNN with the ability to capture higher-level patterns from previous convolutional layers. Again, max-pooling layers are added to reduce the growing dimensionality of the data output from the convolutional layers. Finally, a flattening layer is used as an interface between the convolutional layers and the subsequent dense layers. Dense layers represent the conventional type of neural network. Layer configurations followed from what was commonly used in the research [4, pp.545-547] and [3, pp.392-393] where convolutional layers were set to use 3x3 filters, max-pooling layers used 2x2 filters, and all activation functions were of the "ReLU" type [3, p.258]. The final layer was different from the others in that it used a "Softmax" activation function to provide a confidence value for multiclass classification [3, p.268]. Experimentation was then performed by varying the sizes of the layers and noting the performance of the CNN's training accuracy and validation accuracy.

Tensorflow 2 and Keras

The CNN was created using the Tensorflow 2 module and its Keras wrapper available to Python. Tensorflow allows the neural network developer to conveniently enclose training samples and their class descriptions into objects called "datasets." Furthermore, testing data is also encapsulated into Tensorflow dataset objects. With training and test data ready, the CNN "model" is created layer-by-layer using Keras. Next, the layers are compiled along with the choice of optimizer, loss function, and the metric to optimize. Finally, a fitting function is called against the model object with the dataset objects

and number of epochs as arguments. Tensorflow then proceeds to train the CNN, reporting the training accuracy and validation accuracy after every epoch. Tensorflow also provides training and validation process configurations called “pipelines” that can be used for data shuffling, caching, batching, prefetching, and most important of all: preprocessing. After configuration and fitting, a trained Tensorflow model is available to save to disk in hdf5 format.

Tensorflow 2 Model for Galaxy Classification

The following figure represents Tensorflow 2’s summary of the final CNN model developed for this project.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 200, 200, 1)       0

conv2d (Conv2D)              (None, 198, 198, 128)     1280

max_pooling2d (MaxPooling2D  (None, 99, 99, 128)       0
)

conv2d_1 (Conv2D)            (None, 97, 97, 128)       147584

max_pooling2d_1 (MaxPooling  (None, 48, 48, 128)       0
2D)

conv2d_2 (Conv2D)            (None, 46, 46, 128)       147584

max_pooling2d_2 (MaxPooling  (None, 23, 23, 128)       0
2D)

flatten (Flatten)            (None, 67712)             0

dense (Dense)                (None, 256)               17334528

dense_1 (Dense)              (None, 256)               65792

dense_2 (Dense)              (None, 128)               32896

dense_3 (Dense)              (None, 3)                 387

=================================================================
Total params: 17,730,051
Trainable params: 17,730,051
Non-trainable params: 0
_________________________________________________________________
```

Figure 4 Tensorflow 2 model summary for the Galaxy Classifier App

Following the general consensus of the research literature [4, p.547] and [3, p.393], the following Tensorflow 2 arguments were passed to the compile() function of the model object:

- Optimizer = ‘adam’
- Loss = ‘sparse_categorical_crossentropy’

- Metrics = [‘accuracy’]

Epoch count was limited to an initial run of 30. If the training accuracy suggested that the CNN’s accuracy had more potential for improvement, an additional 30 epochs were run.

Additional minor preprocessing was performed by the Python application. This included cropping a 200x200 pixel image from the original 424x424 image and a normalization of the pixel values from the original 0-to-255 range (8-bit unsigned integer) to a 0-to-1 range (32-bit, floating-point).

The preprocessed images were used to the fullest extent (8,440 out of 8,443 available) with an 80% allocation to the training dataset (6,752 images) and a 20% allocation to the validation dataset (1,688 images). Once all training epochs completed, the Python application (main_final.py) plotted the training accuracy and the validation accuracy as a function of the epoch number.

Results

The final model was trained and validated over a total of 60 epochs in two, 30-epoch runs. The first run resulted in the data shown in figure 5.

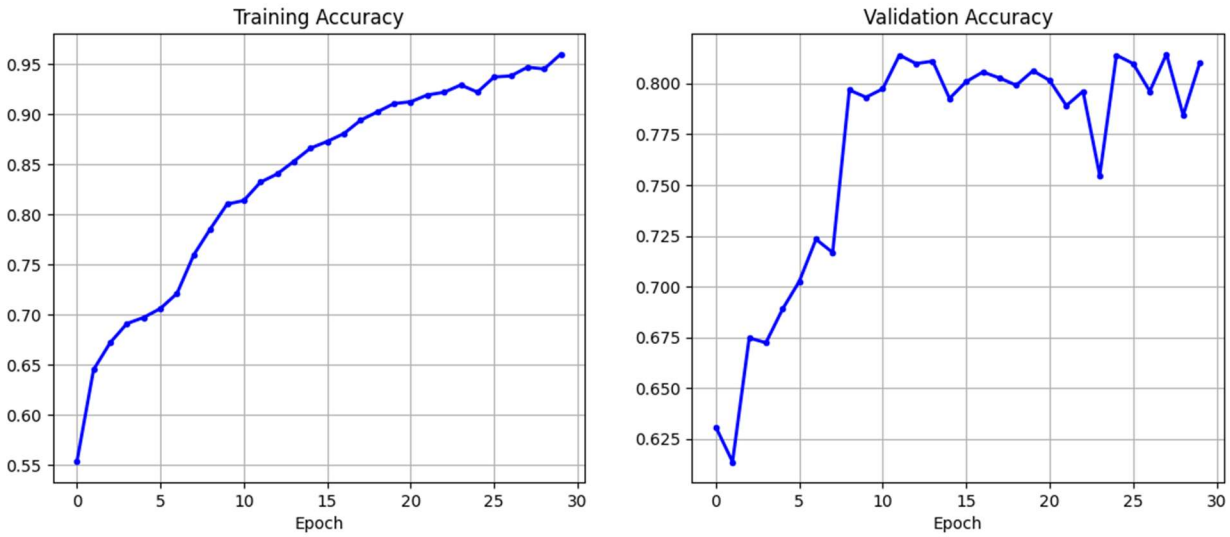

Figure 5 Training and validation accuracy of the final model, first 30-epochs

Training accuracy yielded 96% with validation accuracy yielding 81%. In an attempt to improve the validation accuracy, a second, 30-epoch training run was done yielding the following result in figure 6.

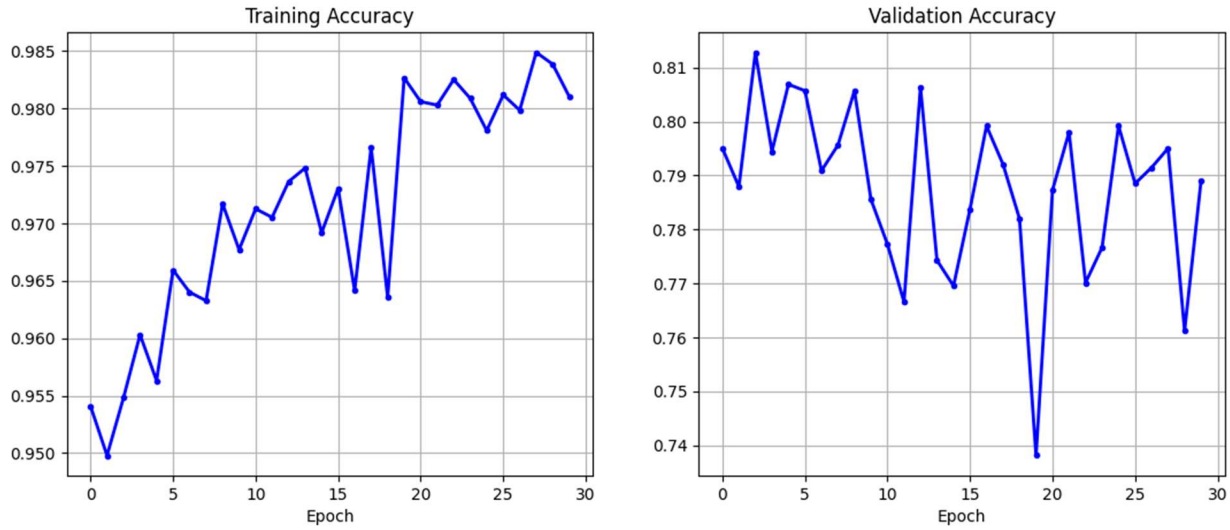

Figure 6 Training and validation accuracy of the final model's second, 30-epoch run.

Here, training accuracy improved to 98%, but validation accuracy declined to 79%, an indication that the additional training was tending toward overfitting the data.

The final realization of the Python program (main_final.py) is an interactive graphical user interface (GUI) shown in figure 7.

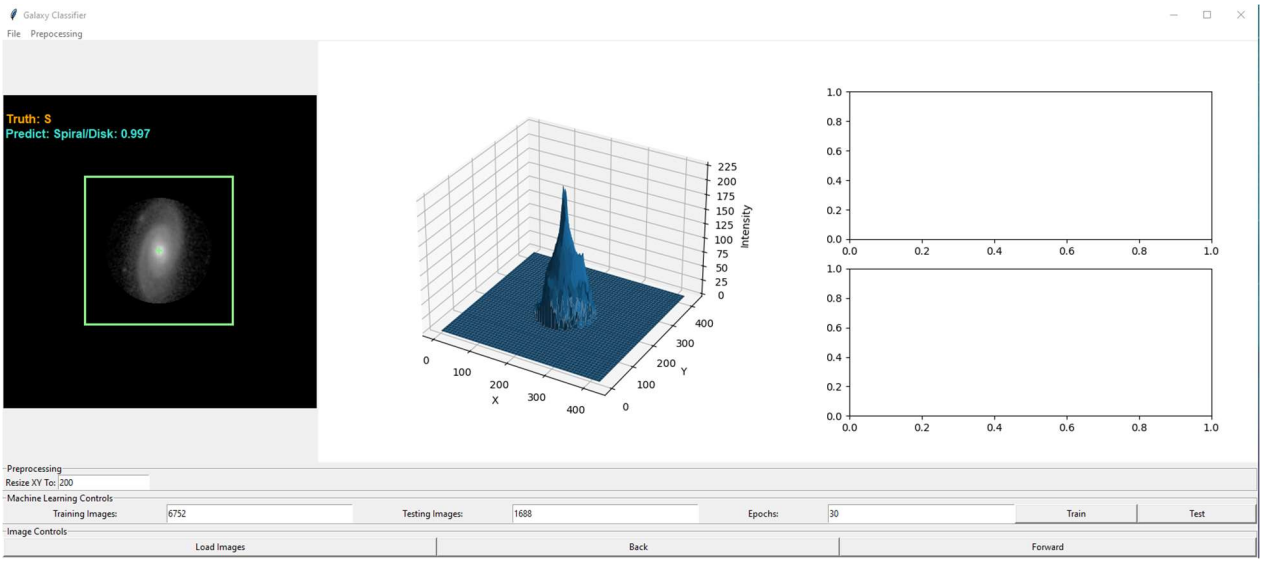

Figure 7 The Galaxy Classifier application GUI

This GUI allows the user to observe the preprocessed galaxy image, along with its true and predicted classification. The 3D plot in the center provides a topographical perspective of the image pixel intensity. Controls are provided at the bottom to load the image set, cycle through the images, and train the CNN.

Conclusions

Machine learning with convolutional neural networks most-certainly demonstrates a promising capability for the classification of galaxies in astronomical surveys. High training accuracies are demonstrably obtained with the convolutional neural network employed, but high-validation accuracies are a bit more challenging to achieve. The use of custom-developed preprocessing methods, such as the technique demonstrated in this project, contribute significantly to the CNN's ability to obtain high-accuracy classification predictions. With the freely-available dataset provided by Galaxy Zoo, the free tools available to Python, any motivated machine-learning developer who wishes to test their own preprocessing algorithm and neural network tuning skills can undertake this galaxy classification challenge.

References

[1] https://zenodo.org/record/3565489#.ZCt2rPbML-h

[2] Kulkarni, Arun D. "Artificial Neural Networks for Image Understanding." Van Nostrand Reinhold, 1993.

[3] Liu, Yuxi H. "Python Machine Learning by Example." Packt Publishing, 2020.

[4] Raschka, Sebastian and Mirjalili, Vahid. "Python Machine Learning." Packt Publishing, 2019.

[5] Willet, Kyle W., et al. "Galaxy Zoo 2: Detailed Morphological Classifications for 304,122 Galaxies from the Sloan Digital Sky Survey." Monthly Notices of the Royal Astronomical Society, 2013.

Acknowledgements

Special thanks to Christopher Hoppe for his generous interim review of this project.