Using Convolutional Neural Networks for Hubble Image Classification

by Edward Serafimescu, UCLA '26

Abstract

The purpose of this program is to test the capabilities of convolutional neural networks on image classification as well as experiment with image tensor transforms using techniques to blur, cut, invert, flip, and jitter. Using images taken using the Hubble space telescope, the goal is to correctly classify images into the three categories: solar system, nebulae, and galaxies. The data was downloaded from Kaggle, sourced from NASA.

1. Introduction

The goal of a convolutional neural network is to break down image patterns across layers of data in a supervised machine learning environment throughout training. Backpropogation ensures that error is minimized using the gradient descent algorithm or similar optimization techniques. In this program, the network is trained to classify images taken by the Hubble space telescope into types of extrasolar phenomenon. The data was taken from the archives powered by Kaggle. To make sure the model functioned properly, the images were all cropped to 224 by 224 pixels around the center. Since the data only consisted of roughly 600 images, the existing images were copied and augmented using `torchvision` library. Effects such as GaussianBlur, Jitter, and various horizontal flips and perspective changes gives the model more opportunities to classify images every when it is less recognizable. Adding these images, the total comes to roughly 1200 images, which will aid the training process. Although this may not be enough; many other networks have relied on much more images. Despite this, the model will attempt to learn from a portion of those images. With data preparation complete, the PyTorch class that dictates the model is next.

2. The Workings of Convolution

Each image in the data will be recognized as an RGB tensor with dimensions along each color axis being the pixel dimensions. Per my algorithm, the data will undergo five convolutions and subsequently five fully connected layers before classification is done.

2.1. Conv2D

Conv2D is a function from the PyTorch neural network library that convolves the image data using a fixed square kernel in two dimensions. This kernel is a $n \times n$ matrix that goes over an image and looks for patterns based on changes in the tensor data. The larger the kernel, the more "big picture" or general the image data will become. For this program, a repeated 3×3 kernel was chosen, similar to AlexNet deep convolution strategy. It is important to keep track of the dimensions of the image tensor data so a helper file calculates the size after each convolution. The ultimate goal of convolution in image processing is to produce a feature map capable of detecting all useful patterns seen in the three possible image classifications.

2.2. MaxPool and ReLU

MaxPool is a tool used to generalize pixel data and to improve general performance. For this program, the pixel of most relevance within a 2×2 space is chosen to represent that section of the image. This may be very handy in condensing large spaces of alike pixels but may cause details to be glossed over, such as thin edges or smaller image features. To prevent the overuse of this feature, it will only be used on the first two and final of the convolutions.

ReLU will be the activation function for this program. It is represented by:

$$\text{ReLU}(x) = \max(0, x) \tag{1}$$

The non-linear nature of the function allows the CNN to handle more complex decision boundaries while avoiding the Vanishing Gradient Problem common to sigmoid or softmax activation. Along every convolution, ReLU will be used.

3. Optimizers

Optimizers are the type of gradient descent that will work to minimize loss following the training process. For this project, SGD and Adam are tested as optimizers.

3.1. SGD

SGD, or Stochastic Gradient Descent, is an altered method of regular gradient descent that relies on the gradients of individual data points, or often batches of data, in terms of the loss function rather than the entire data set. The formula is as follows:

$$w^{t+1} = w^t - \alpha \nabla f_i(w^t) \tag{2}$$

where w is the current position, α is the learning rate, and the gradient of f_i where i is an individual data point or a batch. These calculate w^{t+1}, the next position in the descent. These values create a wildly varying descent path, compared to the

smoothness of regular GD, that eventually approaches the convergence area. The method can be inexact but is more efficient in computation when dealing with large data pools.

3.2. Adam

Adam is another method of descent that focuses on an adaptive learning rate that relies on the exponential average of the squares of gradients. This allows for fast convergence at the cost of a less optimal solution. This would be a good algorithm if error remains consistently high after some time.

4. Running the Neural Network

The model used is made from five convolutions and three linear connectors. All starting images were cropped from the center at 224 pixels and run through several augmentation filters such as jittering and blurring to give the machine an edge with less recognizable images as well as boost the number of image data for training. With each convolution and MaxPool, the images are processed and a feature map is created. The features would then be detected in an image using a fully connected neural network. Lastly, they are classified based softmax of the output vector. A visualization of model architecture can be found below in Figure 1. As shown, the final classifications

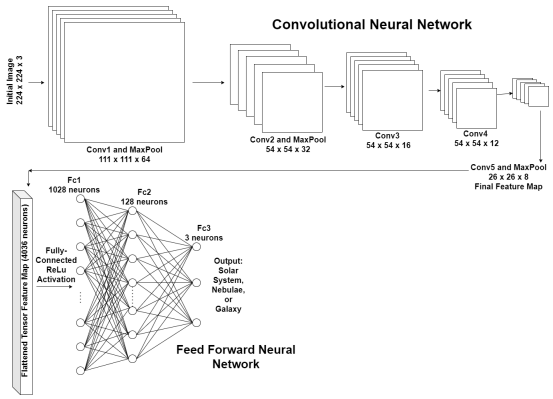

Figure 1: CNN Visualization

will be either solar systems, nebulae, or galaxies.

5. Summary and Conclusions

The first trial was done with SGD with the following: Since

SGD Params	Values
Batch Size	4
Learning Rate	0.001
Epochs	100
Error	≈ 85

Table 1: First Trial: SGD

error remained stagnant, the Adam optimizer replaced SGD to encourage convergence. The prediction among the batches were constant: it thought every image was a solar system. The

Batch Size	32
Learning Rate	0.01
Epochs	500
Error	≈ 75

Table 2: Second Trial: Adam

next attempt went as: Despite changing the optimizer, the error only decreased slightly. This time the machine guessed each image as nebulae. To see if further convergence was possible, the training done by the second trial was built apon using the same parameters but with a smaller batch size of 24. The results were mostly the same. Lastly, a new model was trained with up to 1000 epochs, but the error scarcely moved from 75. The models would not converge.

Despite the model working in theory and without error, proper convergence seemed unlikely given the parameters. This could be because it could take up to 10000 epochs to train the model properly, a process that would be consuming both time wise and on the computer where the program lies. Another possibility is that there is not enough image data, for the 1200 half augmented images pale in comparison to the amount used in AlexNet. For the future, both more Hubble images would be acquired and much more epochs in order for the classification to approach accurate readings.