Improving the identification of strong gravitational lenses using convolutional neural network (CNN)

Ivashkov, Petr

February 15, 2022

1 Introduction

In this work, the convolutional neural network to identify strong gravitational lenses was reproduced, trained and tested on a representative dataset. Further, the performance of the network was assessed and improvement attempts were discussed. Finally, an approach to enforce the invariance of the network to 90° rotations of the input image was suggested and implemented.

2 Basic functions

To be able to work with the model, some crucial functionalities needed to be implemented. First of all, the images from the telescope come in a rather specific format $.fits$, so that a data loader was implemented. For that, the *astropy* library was used, which allows a straightforward conversion of a $fits$-file into a *numpy* array, which can be "fed" to the model. The loaded data is then split into training, validation and test dataset. For the model to be trained on large datasets, it is essential to be able to dump the model on disk and load it afterwards. In this way, the model can be trained in multiple steps without running out of allocated memory. Finally, prediction function of a trained network on a single input image should be available.

3 Data set

The underlying dataset consists of 40 000 positive and 40 000 negative examples of gravitational lensing [2] and was provided by R. Cañameras. This data set was split into training, validation and test set like 56%, 14% and 30% respectively. The test dataset was kept aside and did not participate in training. Only in this manner an unbiased evaluation of the model can be given. The training and validation set were shuffled at each training step and 80% of the shuffled set was used for training and 20% for validation.

4 Network model

4.1 Architecture

The network model (figure 1) was adopted from [1] without architectural changes. The model contains 3 convolutional layers (Keras Conv2D), followed by 3 fully-connected layers. Max-pooling layers with 2×2 kernel sizes and $stride = 2$ were inserted after the first two convolutional layers. After each layer a RelU activation function was used, with exception for the single output neuron, where sigmoid activation was applied. The output of the network is a floating point number in range [0,1] and corresponds to the network lens or nonlens prediction. The described architecture results in approximately 27 mio trainable parameters.

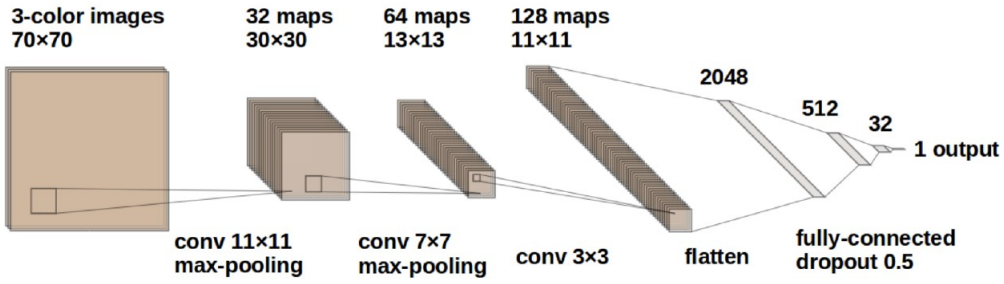

Figure 1: Model architecture

4.2 Configuration

Configuration	Value
Loss function	binary_crossentropy
Batch size	128
Number of epochs	35
Learning rate	$6 \cdot 10^{-4}$
Weight decay	L2 with $\lambda = 10^{-4}$
Optimiser	adam

Figure 2: Model configurations

Most network configurations were as well adopted from the model described in [1] and are listed in table 2. For instance, a dropout of 0.5 was used before the first fully connected layer. This is an efficient technique that consists of randomly ignoring a fraction of neurons (in this case 50%) in order to reduce overfitting on the training set [1]. During the training, the *adam* optimiser was used, as it showed a higher performance compared to the SGD described in [1]. During the training process early stopping was applied at epoch 33, that matched the lowest loss over the cross-validation runs.

4.3 Assessing performance

After training the model on the entire dataset with the described configurations, the 98% accuracy was achieved on the test set. To assess, if one can achieve even better accuracy, two approaches were tried. First, adding or removing one convolutional layer at different positions in the architecture appeared to decrease or not affect the performance on the test set. Second, tuning the model to have a 2 neuron output with a categorical_crossentropy loss function decreased the accuracy of the model. This observation was made by R. Canameras [1] as well and proves the model to have an optimal architecture. At this point, is was not further attempted to increase the accuracy and focus moved to improving the rotational invariance of the model.

5 Improving rotational invariance

5.1 Problem and objective

Convolutional neural networks show a good invariance to spacial translations of the image, but are not rotationally invariant. Precisely it means that feedforwarding the same image input under different rotations results in different network outputs. To assess the deviation of the network on rotated images, 4 prediction are made for the same image under 90° rotations. Average standard deviation within 4 rotations on test set is 4% for the described model, whereby the deviation on a single image can reach up to 30%. Hence my objective was to improve the rotational invariance without significantly decreasing the accuracy.

5.2 Approach

I have considered two approaches to tackle the problem. First idea would be to apply rotations to the input images and train the model on these inputs as well. This is a common data augmentation technique, which

although could result in the network learning many redundant parameters, which are learned independently for each rotation [3]. Another approach would be to apply rotations to the convolution filters. Precisely this means simultaneously feeding several rotated versions of the input image to the convolutional layer and taking the maximum value across these convolutions as a result. This was done by introducing a new *rotational* layer, which incorporates an (overloaded) Conv2D Keras layer 3 and a max-pooling operation. The convolution is calculated for the image rotated at 0°, 90°, 180° and 270° and the bias is added to the maximum value. The overloaded layer is shown in figure 3. The algorithm has no claim to be efficient and was developed as a proof of concept. For example, one could also think of applying rotations to the kernels instead of the input array.

```
class RotationalConv2D(layers.Conv2D):
    def call(self, inputs):
        r0 = self.convolution_op(
            rot90(inputs, k=0) , self.kernel)   # 0° rotation
        r90 = self.convolution_op(
            rot90(inputs, k=1) , self.kernel)   # 90° rotation
        r180 = self.convolution_op(
            rot90(inputs, k=2) , self.kernel)   # 180° rotation
        r270 = self.convolution_op(
            rot90(inputs, k=3) , self.kernel)   # 270° rotation

        # result := maximum output within rotation group
        result = maximum(maximum(r0,r90),maximum(r180,r270))

        if self.use_bias:
            result = result + self.bias
        return result
```

Figure 3: Rotational layer

5.3 Result

In the described approach, the network is explicitly enforced to be invariant to rotations, as the maximum value is always taken as a result. As expected, the new model (trained on the same dataset) predicts the same scores for the rotated version of the same image. At first, the rotational layer was inserted before the first convolutional layer, which slightly decreased the accuracy of the model - from 98% to 95%. Afterwards, the rotational layer was inserted in place of the first convolutional layer with the same kernel size and number of filters. This brought the accuracy back close to 98%. In general, the new network shows very similar accuracies to the old one, given they were trained on the same dataset.

6 Outlook

Introducing a rotational layer has proven to enforce the invariance to 90° rotations. To go even further, one could apply more rotations to the input images - for example rotate the image at 45° each time instead of 90°. In this case, although, one has to deal with the pixels being cropped out of the image due to rotation. A possible solution would be to only use the pixels within a circle circumscribed in the square filter. To further improve the performance, one could think of different network architectures (e.g., ResNet instead of CNN). For instance, a higher accuracy in identifying real-life gravitational lenses was achieved by R. Canameras et al in [2] using the residual neural network.

References

[1] R. Cañameras , S. Schuldt , S. H. Suyu , S. Taubenberger , T. Meinhardt , L. Leal-Taixé , C. Lemon , K. Rojas , E. Savary
HOLISMOKES – Identifying galaxy-scale strong gravitational lenses in Pan-STARRS using convolutional neural networks
URL: https://www.aanda.org/articles/aa/pdf/2020/12/aa38219-20.pdf

[2] R. Cañameras , S. Schuldt , Y. Shu , S. H. Suyu , S. Taubenberger , T. Meinhardt , L. Leal-Taixe , D. C.-Y.Chao , K. T. Inoue , A. T. Jaelani , A. More
HOLISMOKES – New galaxy-scale strong lens candidates from the HSC-SSP imaging survey.
URL: https://arxiv.org/pdf/2107.07829.pdf

[3] Diego Marcos , Michele Volpi , Devis Tuia
Learning rotation invariant convolutional filters for texture classification.
URL: https://arxiv.org/pdf/1604.06720.pdf