Fakultät für Physik
Fakultät für Physik
Technische Universität München

Identifying strong gravitational lenses using CNN

Adivsor: Anton Riedel and Raoul Canameras

Ivashkov Petr

Wintersemester 2021/22

Outline

Inhaltsverzeichnis

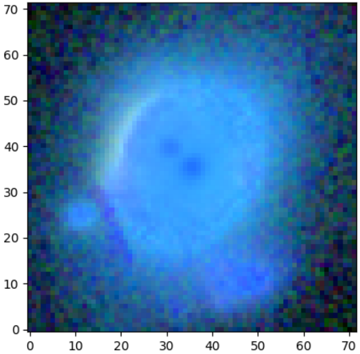
Basic functions

- Load data in FITS format
- Split data into training, validation and test set
- Train and test the network performance
- Dump on disk and load the trained network
- Predict the result on a singular image

Inhaltsverzeichnis

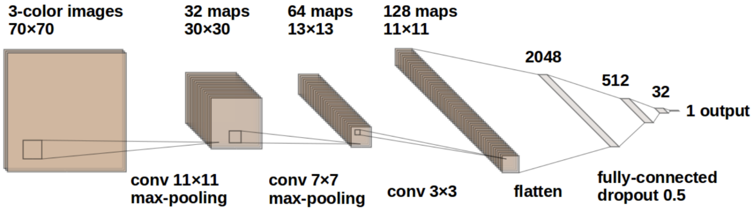
Data-set

- 40K positive (artificial) and 40K negative examples
- Split into training, validation and test set like 56%, 14% and 30% respectively
- Training and validation shuffled with $validation\_split$ = 0.2
- Test data as a separate set



Inhaltsverzeichnis

Network model



- **relu** activation between the layers
- **sigmoid** on the output neuron
- **Total** of around 27M parameters

Inhaltsverzeichnis

Network configurations

- Adopted
  - **loss** = "binary_crossentropy"
  - **batch_size** = 128
  - **epochs** = 35
  - **learning_rate** = 0.0006
  - **Dropout** = 0.5 to prevent overfitting
  - **Weight decay** (applied to kernels) to favor parameters of small magnitude
- Improved
  - **optimizer** = «adam»
  - **shuffle** = True
  - **EarlyStopping** at epoch 33

Inhaltsverzeichnis

Improving the performance

- Changing network hyper-parameters doesn't improve the accuracy significantly
  - Started off with a 2 neuron output and **categorical_crossentropy** loss function
  - **binary_crossentropy** delivers better accuracy
  - Adding and removing one convolutional layer
- Accuracy with adopted hyper-parameters close to 98%

Inhaltsverzeichnis

Rotational invariance

Problem:

- Feedforwarding the same image under different rotations results in different network outputs
- Average standard deviation within 4 rotations on test set around 4%

Objective: Improve rotational invariance without significant decrease in accuracy

Two possible approaches:

- Applying rotations to the input images, i.e. **data augmentation**
- Applying rotations to the convolution filters

Hardcoding rotational invariance

Idea:

- Simultaneously feed several rotated versions of the input image to convolutional layer
- 4 rotations performed 0°, 90°, 270°, 360°

Precisely:

- New rotational layer (overloaded **Conv2D**)
- Convolution is calculated for 4 rotated inputs
- Maximum valued across those convolutions is taken as a **result**
- Finally add **bias**

Note: No claim to be efficient, i.e. just a proof of concept

Rotational layer

```python
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
```

Results and discussion

Results:

- Invariance to local rotations and therefore invariance to global rotations of the input image
- Accuracy close to 95%

What is next:

- More rotations
- Only use the pixels within a circle circumscribed in the square filter - or padding
- Different architectures (e.g., ResNet instead of CNN)
- Data augmentation (e.g. normalize images)