Google Summer of Code

Project Report

Equivalent Neural Network for Dark matter Morphology with Strong Gravitational Lensing

---

Submitted by:

Apoorva Vikram Singh

National Institute of Technology Silchar

Supervisor:

Dr. Emanuele Usai

Anna Parul

Michael Toomey

Pranath Reddy

25th March 2021

1 Problem statement

Equivalent Neural Network for Dark matter Morphology with Strong Gravitational Lensing. The link to repository containing the code for the task can be found here.

2 Dataset Description

The dataset contains grayscale images of size (150, 150). The dataset consists of two different classes. The class sub corresponds to images with dark matter substructure while no_sub corresponds to images without dark matter substructure. The dataset contains a total of 5000 images.

3 Dataset Pre-processing

We cropped the raw images from the dataset to size (128, 128). The images are padded to have a size (129, 129). This allows us to use odd-size filters with stride 2 when downsampling a feature map in the model. In order to reduce interpolation artifacts (e.g. when testing the model on rotated images), we upsample an image by a factor of 3, rotate it, and finally downsample it again.

4 Network Architecture

For this task, I have used Steerable CNNs with E(2)-equivariant convolutions [1]. Classical convolutional neural networks (CNNs) [2] are by design equivariant to translations of their input. This means that a translation of an image leads to a corresponding translation of the network's feature maps. On the other hand, I have used a neural network that is equivariant under all isometries E(2) of the image plane $\mathbb{R}^2$, that is, under translations, rotations and reflections. In contrast to conventional CNNs, E(2)-equivariant models are guaranteed to generalize over such transformations and are therefore more data-efficient. As the network was originally designed for multi-class classification and not binary classification I replaced the last activation layer i.e. the softmax layer with the sigmoid activation layer. I have used PyTorch [3] for implementation. Minute details about the code of the network are mentioned in the Jupyter lab as comments. Given below is the model summary of the network.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
        MaskModule-1          [-1, 1, 129, 129]               0
SingleBlockBasisExpansion-2       [-1, 8, 1, 49]               0
BlocksBasisExpansion-3              [-1, 1, 49]               0
            R2Conv-4        [-1, 192, 125, 125]               0
       BatchNorm3d-5       [-1, 24, 8, 125, 125]              48
    InnerBatchNorm-6        [-1, 192, 125, 125]               0
               ReLU-7        [-1, 192, 125, 125]               0
  SequentialModule-8        [-1, 192, 125, 125]               0
SingleBlockBasisExpansion-9       [-1, 8, 8, 25]               0
SingleBlockBasisExpansion-10      [-1, 8, 8, 25]               0
```

```
SingleBlockBasisExpansion-11        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-12        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-13        [-1, 8, 8, 25]                0
BlocksBasisExpansion-14              [-1, 192, 25]                0
           R2Conv-15          [-1, 384, 125, 125]                0
      BatchNorm3d-16        [-1, 48, 8, 125, 125]               96
   InnerBatchNorm-17          [-1, 384, 125, 125]                0
             ReLU-18          [-1, 384, 125, 125]                0
 SequentialModule-19          [-1, 384, 125, 125]                0
PointwiseAvgPoolAntialiased-20  [-1, 384, 63, 63]                0
 SequentialModule-21            [-1, 384, 63, 63]                0
SingleBlockBasisExpansion-22        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-23        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-24        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-25        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-26        [-1, 8, 8, 25]                0
BlocksBasisExpansion-27              [-1, 384, 25]                0
           R2Conv-28            [-1, 384, 63, 63]                0
      BatchNorm3d-29          [-1, 48, 8, 63, 63]               96
   InnerBatchNorm-30            [-1, 384, 63, 63]                0
             ReLU-31            [-1, 384, 63, 63]                0
 SequentialModule-32            [-1, 384, 63, 63]                0
SingleBlockBasisExpansion-33        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-34        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-35        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-36        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-37        [-1, 8, 8, 25]                0
BlocksBasisExpansion-38              [-1, 384, 25]                0
           R2Conv-39            [-1, 768, 63, 63]                0
      BatchNorm3d-40          [-1, 96, 8, 63, 63]              192
   InnerBatchNorm-41            [-1, 768, 63, 63]                0
             ReLU-42            [-1, 768, 63, 63]                0
 SequentialModule-43            [-1, 768, 63, 63]                0
PointwiseAvgPoolAntialiased-44  [-1, 768, 32, 32]                0
 SequentialModule-45            [-1, 768, 32, 32]                0
SingleBlockBasisExpansion-46        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-47        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-48        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-49        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-50        [-1, 8, 8, 25]                0
BlocksBasisExpansion-51              [-1, 768, 25]                0
           R2Conv-52            [-1, 768, 32, 32]                0
      BatchNorm3d-53          [-1, 96, 8, 32, 32]              192
   InnerBatchNorm-54            [-1, 768, 32, 32]                0
             ReLU-55            [-1, 768, 32, 32]                0
 SequentialModule-56            [-1, 768, 32, 32]                0
SingleBlockBasisExpansion-57        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-58        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-59        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-60        [-1, 8, 8, 25]                0
SingleBlockBasisExpansion-61        [-1, 8, 8, 25]                0
BlocksBasisExpansion-62              [-1, 768, 25]                0
           R2Conv-63            [-1, 512, 30, 30]                0
      BatchNorm3d-64          [-1, 64, 8, 30, 30]              128
   InnerBatchNorm-65            [-1, 512, 30, 30]                0
             ReLU-66            [-1, 512, 30, 30]                0
 SequentialModule-67            [-1, 512, 30, 30]                0
PointwiseAvgPoolAntialiased-68  [-1, 512, 26, 26]                0
```

```
      GroupPooling-69              [-1, 64, 26, 26]                   0
            Linear-70                     [-1, 64]           2,768,960
       BatchNorm1d-71                     [-1, 64]                 128
              ELU-72                     [-1, 64]                   0
            Linear-73                      [-1, 1]                  65
           Sigmoid-74                      [-1, 1]                   0
================================================================
Total params: 2,769,905
Trainable params: 2,769,905
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.06
Forward/backward pass size (MB): 604.46
Params size (MB): 10.57
Estimated Total Size (MB): 615.09
----------------------------------------------------------------
```

Listing 1: Network Summary

5 Network Training

For training the network, we have split the data into 90% training and 10% testing. This leaves us with 4500 images to train our network on and 500 images to test it. We use binary cross-entropy as a loss function to train the network. The learning rate is kept constant at 0.00005 for the whole duration of training with a weight decay of 0.00001. We have used Adam [4] as the optimizer for the network. The network is trained for 5 epochs.

6 Results

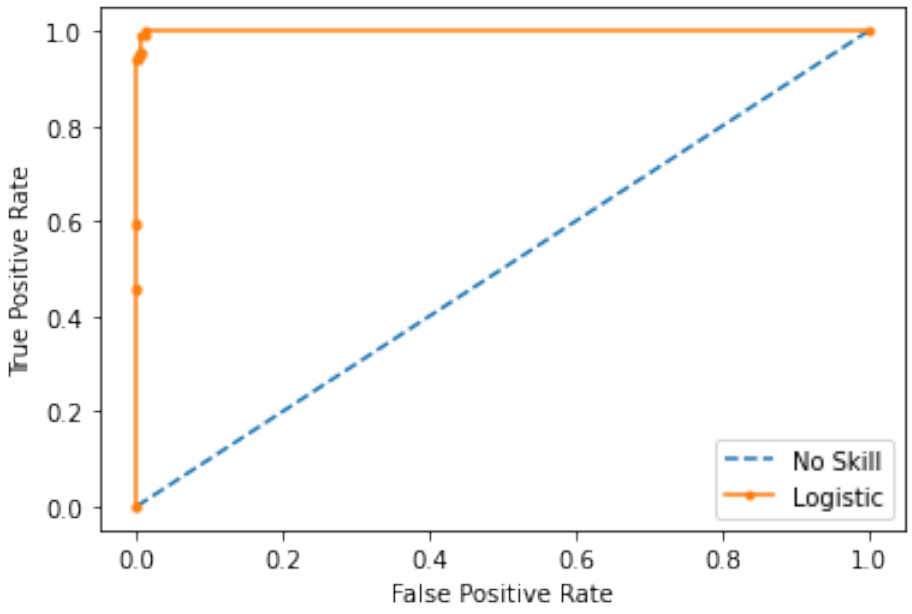

Figure 1: Caption

The proposed model gives an output test accuracy of 99.1% and AUC score of 1.00. The training loss value of binary cross entropy loss function attains a value of 0.047. The AUC curve can be seen in the Fig. 1.

References

[1] M. Weiler and G. Cesa, "General e(2)-equivariant steerable cnns," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.