DeepLense Tests Documentation

ML4SCI, Google Summer of Code

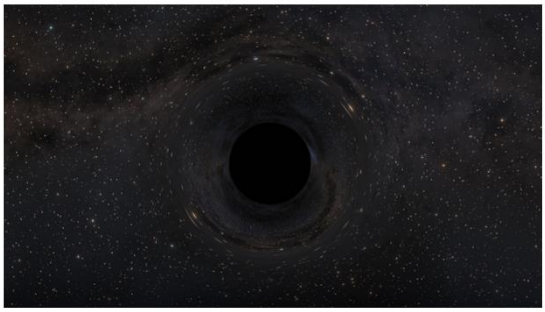
Namritha Maddali

namritha.maddali@gmail.com

Common Test – Classification of Gravitational Lensing Images

https://github.com/namritha-maddali/Classification-of-Gravitational-Lensing

Introduction

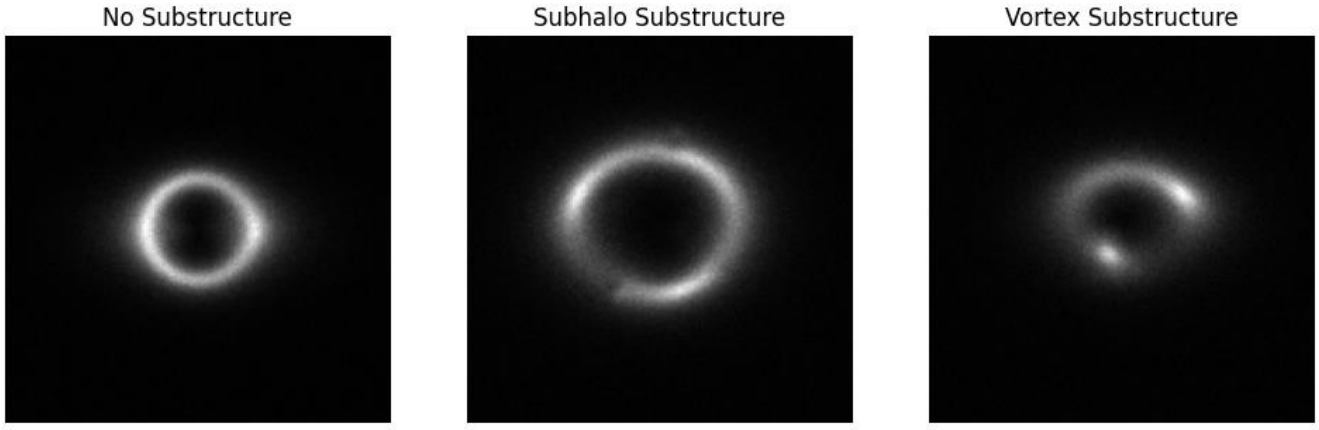


Gravitational Lensing occurs when an extremely massive celestial body causes curvature in spacetime, visibly bending the light around it. Gravitational Lens is the term used to refer to these bodies. Basically, the **light from sources like other galaxies and stars are redirected around the gravitational lens, making the light look "bent"**. A famous example of this can be seen in the theorized black hole images

This was first theorised by Einstein in his General Theory of Relativity, where he described how space around mass concentrations get distorted.

Dataset

- 10,000 images of the class "no"
  - **No Substructure** – smooth and symmetric lensing pattern
- 10,000 images of the class "sphere"
  - **Subhalo Substructure** – small distortions due to subhalos
- 10,000 images of the class "vortex"
  - **Vortex Substructure** – complex, swirling distortions caused by turbulent masses, potentially influenced by strong gravitational interactions or other forces.



Task: Create a classification model to differentiate between the 3 types of gravitational lensing images.

Approach:

Trained a **custom architecture of ResNet** on the data to make predictions

A deep dive into the idea behind custom ResNet

Why ResNet

- Introduces **residual networks**, allowing certain layers to be bypassed.
- This helps in **handling the vanishing gradients problem***
- ResNets are **deeper networks** than typical CNNs and have lesser number of trainable parameters, making it better, faster, and more efficient to train.
- **Incredible training and stable validation loss curves**, ability to generalize and avoid overfitting, and simplicity in architecture make it an ideal model to train for this problem statement!

Why not pre-trained ResNet

- Pre-trained ResNet models are trained on ImageNet's 1k classes, therefore having 1000 neurons as the output of the fully connected layer. We need only three.
- The PyTorch pretrained ResNet model has a lot of restrictions with respect to the image size, number of layers, etc.
- Another reason for experimenting with the core ResNet architecture was curiosity! To explore how it essentially works and how it can be fine-tuned to be solve the task at hand.

The Model

Reference: https://www.digitalocean.com/community/tutorials/writing-resnet-from-scratch-in-pytorch

1. **The Residual Block**
   - It has **2 batch-normalized convolution layers** of size 3x3 each, and is downsampled depending on the stride provided
   - ReLU is applied to the final output of this block.
   - This is the block that is used to **tackle Vanishing Gradients** and skip connections to learn better.

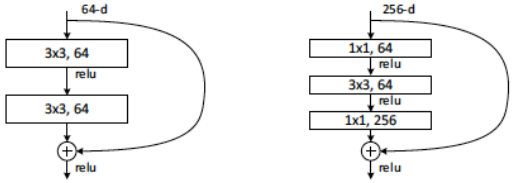
2. **The Main ResNet Architecture**
   - Initial Block = 7x7 Convolution layer with a stride of 2
   - Max Pooling is performed on the initial block to get it to 3x3 size output
   - Residual Layers = 4 Residual blocks with increasing number of planes (filters)
     - First Layer = 3 Blocks
     - Second Layer = 4 Blocks
     - Third Layer = 6 Blocks
     - Fourth Layer = 3 Blocks
   - Adaptive Average Pooling = to downsample the output of the residual layers to a 1x1 form so that it can be fed to the fully connected layer
   - Fully Connected Layer = maps extracted features to the required classes

Finally, it is a ResNet 34 like model with a **total of 34 convolution layers** (6+8+12+6)

```
model = ResNet(ResidualBlock, [3, 4, 6, 3]).to(device)
```

Experimentation with something extra: Bottleneck Block

- The initial (and most accurate so far) version of the architecture used for this task was close to ResNet 34. In attempts to make the neural network deeper, Bottleneck Blocks were experimented with.
- In comparison to ResNet 34's Residual Blocks, this has **3 convolution layers** where the dimensionality of the input is reduced and then restored again. This **reduces the computational overhead** in the second convolution layer (3x3)



- Basically it (right) works similar to ResNet 34 (left) but is much faster.

Observations

- Although introducing the bottleneck blocks made the training very fast, the **convergence rate of the loss observed after each epoch was very low**, that is, the model was not learning much after each epoch.
- The network might be too deep for the given input and task, hence the lower accuracies (maximum achieved was 88%). So, **the classic ResNet 34 Architecture made the main focus**.

```
100%|██████████| 938/938 [01:45<00:00,  8.86it/s]
Epoch [1/50], Average Loss: 1.1474
100%|██████████| 938/938 [01:12<00:00, 12.92it/s]
Epoch [2/50], Average Loss: 1.1243
100%|██████████| 938/938 [00:56<00:00, 16.69it/s]
Epoch [3/50], Average Loss: 1.1188
100%|██████████| 938/938 [00:55<00:00, 16.79it/s]
Epoch [4/50], Average Loss: 1.1140
100%|██████████| 938/938 [00:56<00:00, 16.61it/s]
Epoch [5/50], Average Loss: 1.1128
100%|██████████| 938/938 [00:55<00:00, 16.80it/s]
Epoch [6/50], Average Loss: 1.1099
100%|██████████| 938/938 [00:55<00:00, 16.78it/s]
Epoch [7/50], Average Loss: 1.1104
100%|██████████| 938/938 [00:56<00:00, 16.70it/s]
Epoch [8/50], Average Loss: 1.1090
```

Loss Function: Cross Entropy Loss

$$H(p, q) = -\sum_{x \text{classes}} p(x) \log q(x)$$

(True probability distribution (one-shot); Your model's predicted probability distribution)

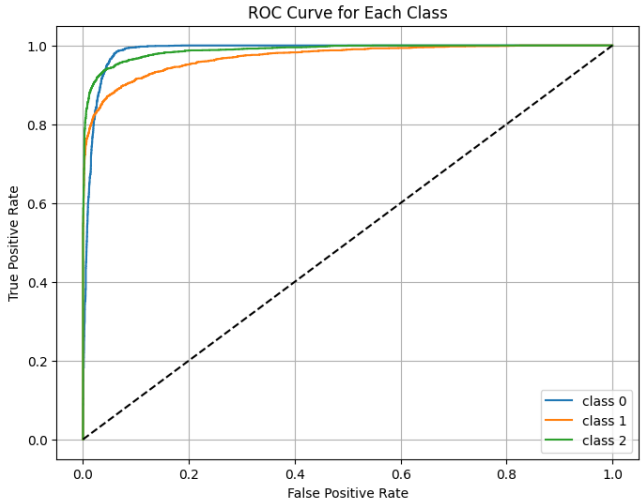
- It is used to measure the classification model's performance by quantifying how likely samples of the dataset are to belong to only one class, hence the ideal value is 0.
- The objective is to reduce the error between predicted and actual values

Optimizer: Stochastic Gradient Descent (SGD)

- This algorithm updates model weights based on the gradient of loss function with respect to the parameters
- Weight Decay = L2 regularization to prevent overfitting
- Momentum = to accelerate learning by smoothing out weight updates (ideal value=0.9)

Metrics:

- **Receiver Operating Curve (ROC):** visual representation of how a model performs across all thresholds.
  - Recall (TPR) = positive data points that are correctly classified as positive
  - Fallout (FPR) = negative data points that were misclassified as positive



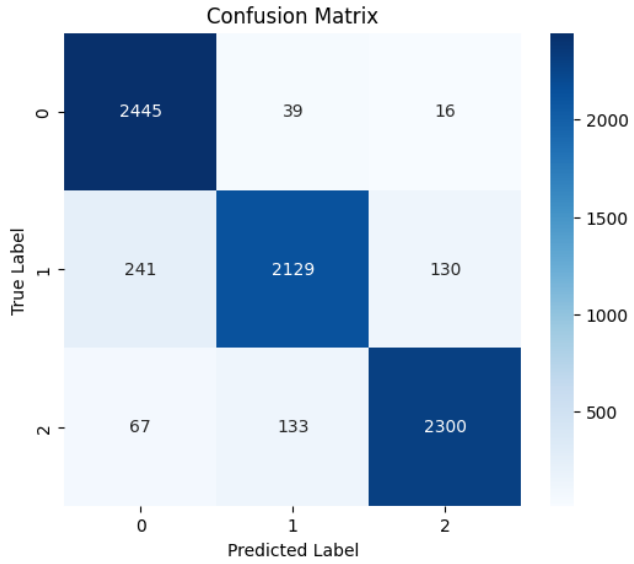
- **Area Under ROC Curve (AUC):** the probability that a randomly chosen positive instance has a higher predictive score (ranks higher) than a negative instance.
  - The AUC values for classes 0 and 2 are the highest so, the model is more effective at correctly ranking positive instances of No Structure and Vortex Structure.

```
AUC Score for Class 0: 0.988
AUC Score for Class 1: 0.970
AUC Score for Class 2: 0.988
```

- **Accuracy:** portion of the data points that have been correctly classified
  - After training the model with the optimal hyperparameters, it gave an accuracy of **91.65%**

```
accuracy = accuracy_score(y_true, y_pred)
accuracy
[30]
... 0.9165333333333333
```

- **Confusion Matrix:** to visualize where the model is going wrong



  - From the confusion matrix we can see that label 1, that is, "sphere" class is the most misclassified out of the other two (accuracy of about 85%)

Conclusion

A ResNet 34 based classification model has been created and it works with an accuracy of about **91.65%**!

The model has been saved as `classification_model.pth` and has been added in the github repository along with the codebase.

References

- https://medium.com/@leonardofonseca.r/a-practical-comparison-between-cnn-and-resnet-architectures-a-focus-on-attention-mechanisms-cee7ec8eca55
- https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
- https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning
- https://stackoverflow.com/questions/57989716/loading-npy-files-as-dataset-for-pytorch
- https://www.digitalocean.com/community/tutorials/writing-resnet-from-scratch-in-pytorch
- https://stackoverflow.com/questions/44429199/how-to-load-a-list-of-numpy-arrays-to-pytorch-dataset-loader
- https://paperswithcode.com/method/bottleneck-residual-block
- https://pytorch.org/tutorials/beginner/saving_loading_models.html
- https://scikit-learn.org/stable/api/sklearn.metrics.html