PHYS4610: Galaxy Zoo - The Galaxy Challenge

Name: Ng Ching Yin
SID: 1155175606

August 22, 2024

1 Galaxy Zoo - The Galaxy Challenge

Galaxy Zoo - The Galaxy Challenge is a Kaggle competition in classifying morphologies of galaxy images. Participants are required to give a weighted probability distributions for 11 set of responses, which are weighted on the probability of the previous questions that led to that set of response (See Table 1).

The final test score is evaluated by root mean square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}, \tag{1}$$

where N is the number of galaxies times the total number of responses, p_i is the predicted probability and a_i is the actual probability obtained by survey. For more detailed information about the competition, please refer to the official website (https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge).

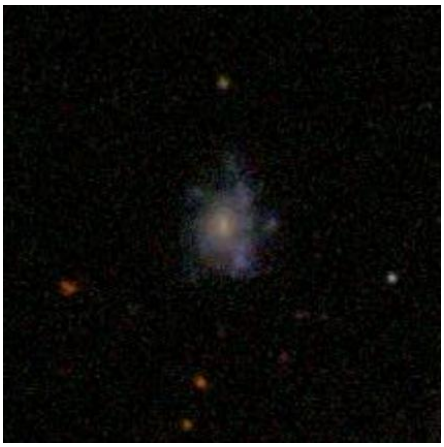

Figure 1: Example of galaxy images

Table 1: The Galaxy Zoo 2 decision tree [1]

Task	Question	Responses	Next
01	*Is the galaxy simply smooth and rounded, with no sign of a disk?*	smooth features or disk star or artifact	07 02 **end**
02	*Could this be a disk viewed edge-on?*	yes no	09 03
03	*Is there a sign of a bar feature through the centre of the galaxy?*	yes no	04 04
04	*Is there any sign of a spiral arm pattern?*	yes no	10 05
05	*How prominent is the central bulge, compared with the rest of the galaxy?*	no bulge just noticeable obvious dominant	06 06 06 06
06	*Is there anything odd?*	yes no	08 **end**
07	*How rounded is it?*	completely round in between cigar-shaped	06 06 06
08	*Is the odd feature a ring, or is the galaxy disturbed or irregular?*	ring lens or arc disturbed irregular other merger dust lane	**end** **end** **end** **end** **end** **end** **end**
09	*Does the galaxy have a bulge at its centre? If so, what shape?*	rounded boxy no bulge	06 06 06
10	*How tightly wound do the spiral arms appear?*	tight medium loose	11 11 11
11	*How many spiral arms are there?*	1 2 3 4 more than four can't tell	05 05 05 05 05 05

2 Training a Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) are known to be effective in image classification tasks. In this section, I will briefly explain the process of training a CNN model in PyTorch. Source code can be found in the GitHub repository: https://github.com/alvinng4/GalaxyZooChallenge

2.1 Data Preprocessing

The training dataset consists of 61578 galaxy images in $(424, 424)$ pixel size and their probability distributions. For the training images, we apply the following transformation:

```
transform_train = transforms.Compose([
    transforms.RandomRotation(360),
    transforms.CenterCrop([256, 256]),
    transforms.Resize([128, 128]),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
])
```

Since rotation and reflection of galaxy images does not affect the concerned properties, we are free to rotate and flip the images randomly to augment the training dataset. The images are cropped to a pixel size $(256, 256)$, resized to $(128, 128)$, and normalized to standardize the pixel values.

2.2 Model Architecture

The model consists of 4 convolutional layers and 3 fully connected layers, with approximately 20 millions trainable parameters. In the convolutional layers, I used a kernel size of 17, 13, 9 and 7 respectively. They all followed by a batch normalization layer, a ReLU activation function and a pooling layer (Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ Pooling). Batch normalization layers are used to speed up the training and to provide regularization.

For the output, a sigmoid layer are used to compute the probability. Originally, Softmax and weighting layers were used to calculate the weighted probability distributions. However, they did not perform well in practice.

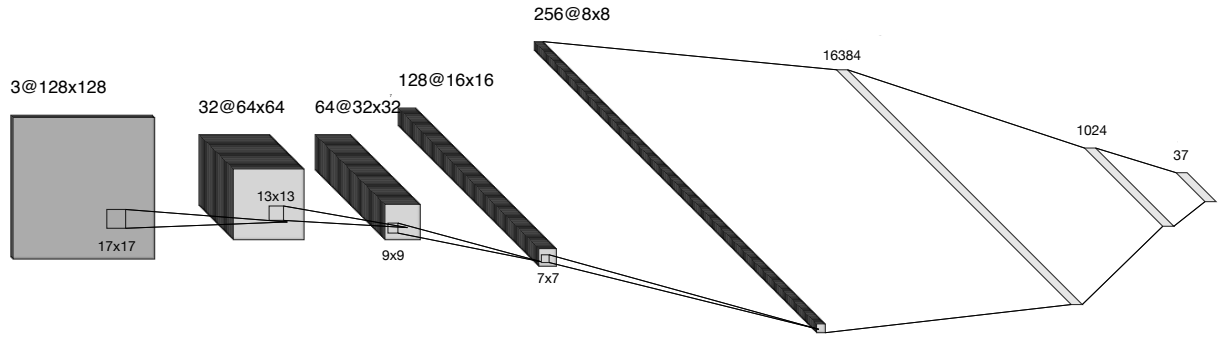

Figure 2: illustration of the model architecture

2.3 Training details

The model is trained with the Adam optimizer and binary cross entropy loss function (`torch.nn.BCELoss`). I also tried the mean squared error loss function but the difference is negligible.

When choosing the batch size, I noticed that the training dataset is highly imbalanced. If we convert the probability distributions of class one into one-hot labels, we can see that class 1.3 is underrepresented (Class 1.1: 26693 Class 1.2: 34826 Class 1.3: 59). Therefore, this provides a starting point for choosing the batch size. At the end, a batch size of around 100 is adopted such that images from class 1.3 appears every 10 batches.

For the learning rate scheduler, I chose `CosineAnnealingWarmRestarts` from PyTorch, which allows the model to escape local minima by restarting the learning rate and slowly converge to the minima by cosine annealing. The initial learning rate is set to 0.0001 with the following parameters, and the model is ran for 240 epochs:

```
CosineAnnealingWarmRestarts(optimizer, T_0=60, T_mult=3, eta_min=1e-8)
```

The run time for each epoch is about 90 seconds, and therefore takes about 7 hours to complete the training.

2.4 Results

During initial training, we split the training, validation and test set with a ratio of 0.85 : 0.05 : 0.10. The test result is as follows:

```
RMSE: 0.07233613375313488

Class one:
              precision    recall  f1-score   support   Confusion matrix:
           0       0.84      0.89      0.87      1869   [[1658  211    0]
           1       0.91      0.87      0.89      2438    [ 306 2132    0]
           2       1.00      0.50      0.67         4    [   0    2    2]]

    accuracy                           0.88      4311
   macro avg       0.92      0.75      0.81      4311
weighted avg       0.88      0.88      0.88      4311
```

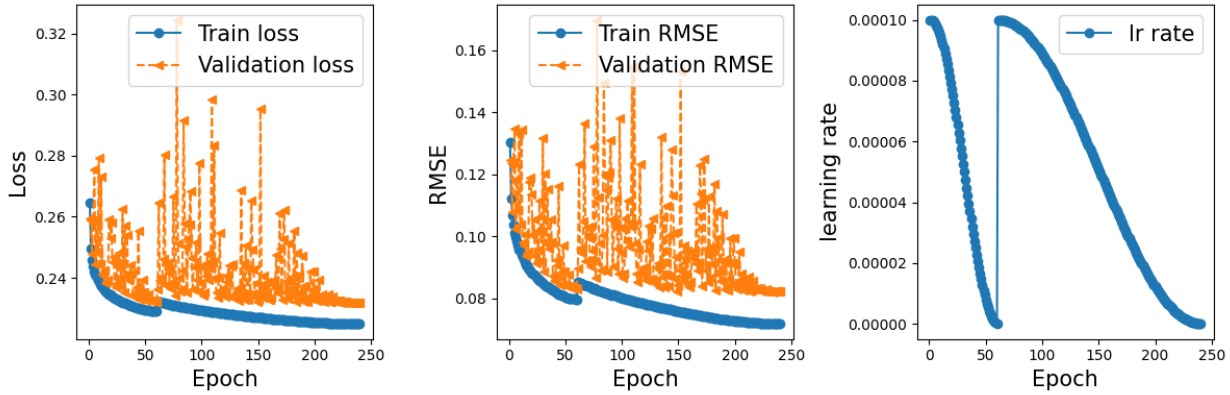

Figure 3: Training the model with validation dataset

After obtaining a satisfactory result, we trained a new model with the full training dataset. Then, we processed the test dataset from Kaggle and submitted the predictions. The final score from the best model is 0.08003 (cf. winner: 0.07491).

In addition, we used AblationCAM to visualize the convolutional layers (See Figure 4) by passing a galaxy image through the first two convolutional layers. We can see that the first layer captures the center of the galaxy (local features), and the second layer captures the whole galaxy (global features). Note that the visualization result are found to varies greatly from run to run.

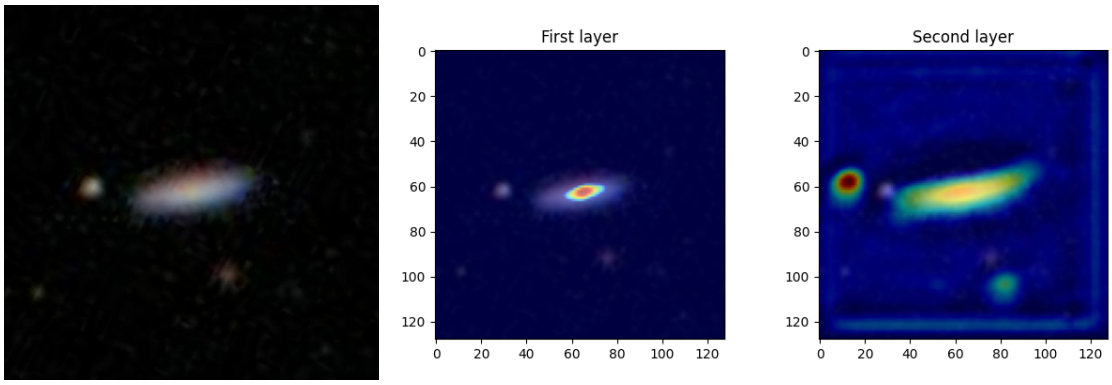

Figure 4: Visualization of the convolutional layers using AblationCAM from the pytorch-grad-cam package

2.5 Comments

The model went through many testing and changes, which led to the following observations:

1. Using dropout layers reduced the performance of the model.

Dropout layers are added in different positions of the model with a 0.2 dropout rate. Yet, this caused a decrease in performance. It is possible that our model is too shallow to benefit from dropout layers, and batch normalization layers have provided enough regularization.

2. The model benefits from a large kernel size in the convolutional layers.

It is known that convolution is a local operation that represents the receptive field of the model. It is hoped that after several convolutional layers, the model can capture the global features of the image. Initially, the kernel size is set to 7, 5, 3, 3 for the four layers respectively, and received a score of 0.091 from Kaggle. However, after a simple adjustment to 17, 13, 9, 7, the score improved significantly to 0.082.

3 Improving the CNN model

3.1 Convolutional Block Attention Module (CBAM)

To further improve our model, we can include self-attention mechanism, which allows the model to focus on important features. Here, we adopt the Convolutional Block Attention Module (CBAM) proposed by Woo et al. [2]. It consists of a channel attention module (CAM) and a spatial attention module (SAM). Below are some figures from the original paper that gives an overview of CBAM.

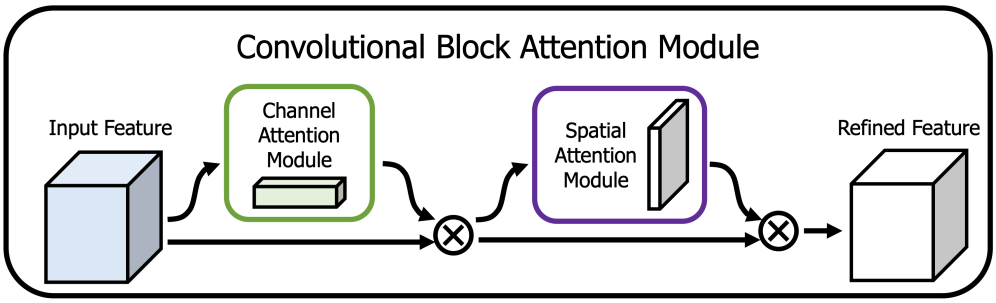

Figure 5: Overview of the Convolutional Block Attention Module (CBAM) [2]

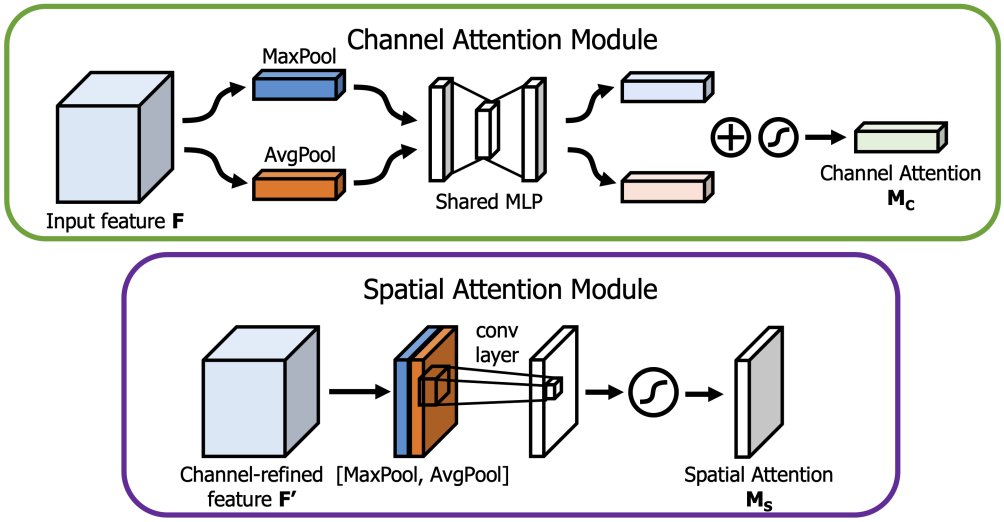

Figure 6: Overview of the channel attention module (CAM) and spatial attention module (SAM) [2]

Now we can simply insert CBAM into our convolutional layers:

CNN: (Conv → BatchNorm → ReLU → Pooling)

CNN-CBAM: (Conv → BatchNorm → ReLU → CBAM → Pooling)

In CAM, we used a reduction ratio $r = 1$ for the shared MLP, and in SAM, we used a kernel size of 7 for the convolutional layer. We obtain a test score of 0.07972 (cf. CNN: 0.08003).

3.2 Model averaging

During development, we have created many variations of the CNN model. By averaging the predictions of these models, we can reduce the variance and improve the accuracy of the predictions. Below are the scores of our best 10 models, and the scores by model averaging. We can see that by averaging our top 8 models,

we achieved a score of 0.07716, which has an improvement of 0.00256 from the best individual model. This brings us to the 2nd place in the competition.

Table 2: Best scores of individual models

Number	Scores
1	0.07972
2	0.08003
3	0.08026
4	0.08037
5	0.08097
6	0.08102
7	0.08111
8	0.08114
9	0.08116
10	0.08168

Table 3: Scores by model averaging

Number of models	Scores
1	0.07972
2	0.07816
3	0.07775
4	0.07754
5	0.07736
6	0.07726
7	0.07726
8	**0.07716**
9	0.07722
10	0.07719

4 References

[1] W. Willett, J. Lintott, P. Bamford, et al., 2013, "Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey", *MNRAS* 435: 2835-2860.

[2] S. Woo, et al., "CBAM: Convolutional Block Attention Module", *European Conference on Computer Vision*, 2018