Classifying galaxy morphologies with Convolutional Neural Networks and Vision Transformers

Ching-Yin Ng

Department of Physics, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong, China

(Dated: November 30, 2024)

Abstract

The Galaxy Zoo project provided a large amount of morphological classifications for galaxy images through the help of volunteers. However, this approach required a vast amount of resources and time. In this paper, we explore the use of Convolutional Neural Networks (CNN) and Vision Transformers (ViT) to automate the classification process. We first introduce Convolutional Neural Networks by constructing a baseline model from scratch. Then, we attempt to improve its performance by adding self-attention modules. Finally, we compare the performance of vision foundation models: ResNet, ConvNeXt and Dinov2. All source codes are available at the GitHub repository: https://github.com/alvinng4/GalaxyZooChallenge.

INTRODUCTION

The Galaxy Zoo project [1] [2] provided morphological (the visual appearance or shape) classifications for millions of galaxies. This was made possible by combining classification responses from hundreds of thousands of volunteers online. However, this kind of approach requires vast amount of resources and time. Thus, machine learning is a compelling solution that enables automation of the process.

Convolutional neural networks (CNN) are known to be effective in image classification tasks. In this paper, we first introduce CNN by constructing a baseline model from scratch. Using data augmentation techniques combined with self-attention modules, we attempt to improve the performance of the baseline model. Finally, we explore and compare the performance of vision foundation models: ResNet, ConvNeXt and Dinov2.

The source codes can be found at the GitHub repository: https://github.com/alvinng4/GalaxyZooChallenge

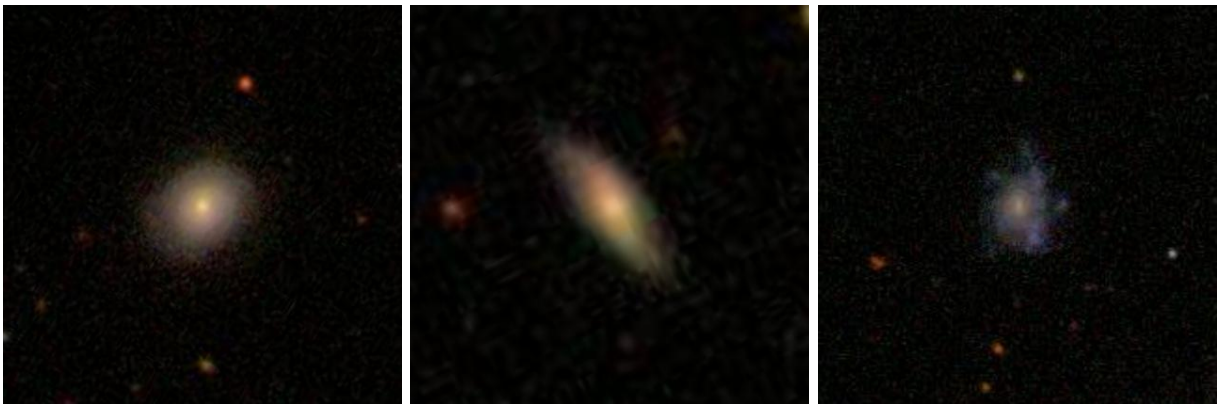

FIG. 1. Galaxy images from the Galaxy Zoo competition dataset

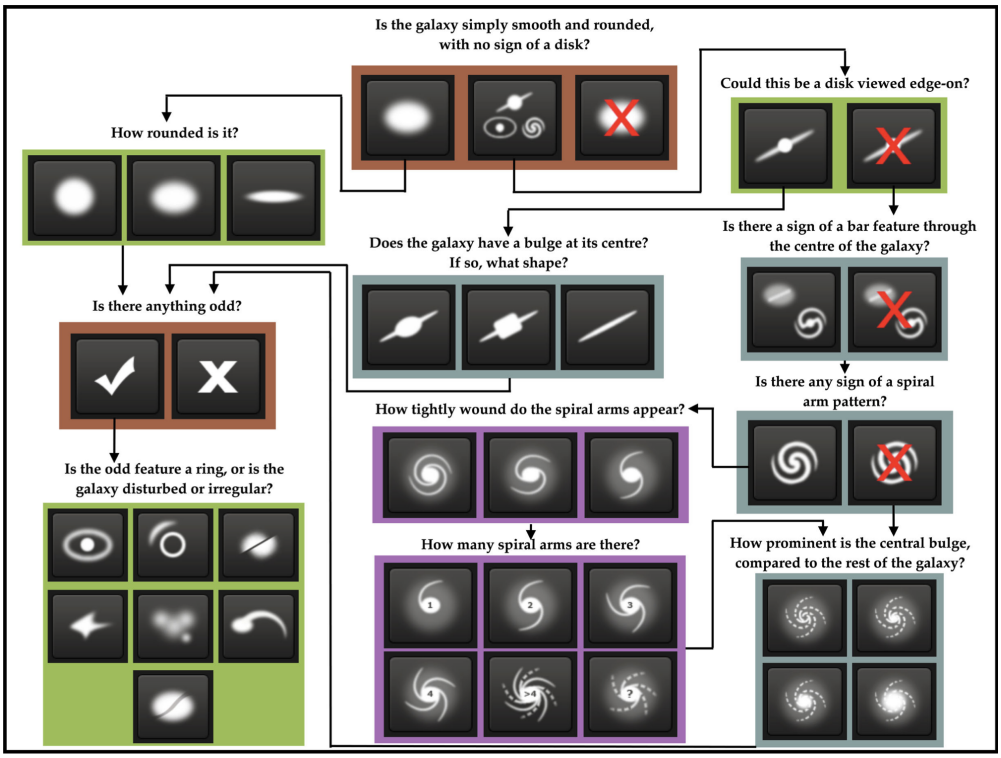

FIG. 2. Flow chart of the decision tree from the Galaxy Zoo 2 paper [2]. It begins at the top centre. Tasks outlined in brown are asked of every galaxy. Tasks outlined in green, blue and purple are (respectively) one, two or three steps below branching points in the decision tree.

METHODS

Galaxy Zoo - The Galaxy Challenge is a Kaggle competition held in 2014 [3]. Contestants were given a set of galaxy images to predict a weighted probability distributions of responses from 11 questions using machine learning methods. Figure 2 shows a flow chart that explains the decision tree of responses.

Through this competition, we are able to obtain a standard dataset and a evaluation score to compare the performance of different models.

Evaluation metric

The 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 galaxy images times the total number of responses per image, p_i is the predicted probability and a_i is the target probability obtained from volunteers.

The winner of the competition, S. Dieleman, achieved a best single model score of 0.07693 (private leaderboard), and a score of 0.07492 (private leaderboard) after model averaging of 18 models [4]. In this paper, we focus on the single model score since model averaging requires large amount of computational resources.

Data Preprocessing and Augmentation

The training dataset consists of 61578 galaxy images in $(424, 424)$ pixel size and their probability distributions. We split the dataset into 9:1 for training and validation. Before feeding the images to the model, we cropped and resized the images to $(128, 128)$ for our baseline models and $(224, 224)$ for the vision foundation models, and then normalized the image with mean and standard deviation equals to 0.5 to standardize the pixel values.

Notice that the morphologies of galaxy images are rotational and reflectional invariant. Therefore, we are free to rotate and flip the images to augment the training dataset. Moreover, zooming in or out the images should also not affect the morphologies. So, we randomly pick $z \in [184, 352]$ for our baseline models and $z \in [128, 282]$ for the vision foundation models as the cropping size before resizing to create different scaling of an image.

Augmentation of test dataset

In training, we exploited rotational, reflectional and scaling invariances to augment the dataset. Here, we can also make use of them to reduce variances of the output. After training a model, we created a combination of rotated images in 8 angles and horizontally flipped images from the test dataset, together forms $8 \times 2 = 16$ images per original image. Then, they were passed to the model to produce 16 individual predictions. At last, we average all the predictions to produce a final prediction.

Scaling the image was also found to improve the final result. However, producing many predictions takes a lot of time, so we did not include scaling to reduce computational cost.

Training details

All models were trained on the PyTorch framework. The Adam optimizer and Mean Square Error Loss function were used to train the models, with a batch size of 105. For the learning rate scheduler, a simple cosine annealing scheme was chosen, with initial learning rate set to 0.0001, as shown in the right of figure 3. For the number of epochs, we experiment with different values and chose the number that gave the best validation score.

As for the hardware, we used a single AMD 5975WX (64 cores) CPU and RTX A6000 (48GB) GPU.

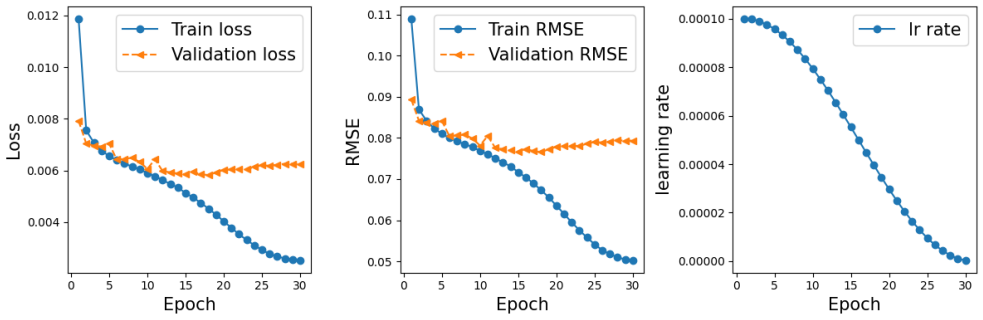

FIG. 3. Training Dinov2 model for 30 epochs. The loss, RMSE and learning rate of the training and validation set are shown. After 15 epochs, although the training RMSE continues to improve rapidly, the validation RMSE starts to increase. This shows the importance of selecting the right number of epochs to prevent overfitting.

BASELINE MODEL

Basics of Convolutional Neural Networks

Convolution is a local operation that acts on a small patch of an image. Therefore, it allows us to obtain the local features of an image, such as edges, corners and textures. As a result, a convolutional layer represents the local receptive field of the model. With several convolutional layers combined with downsampling layers, the model can effectively extract global features from an image. Finally, fully connected layers can be used to map the extracted features to the target probabilities.

Model Architecture

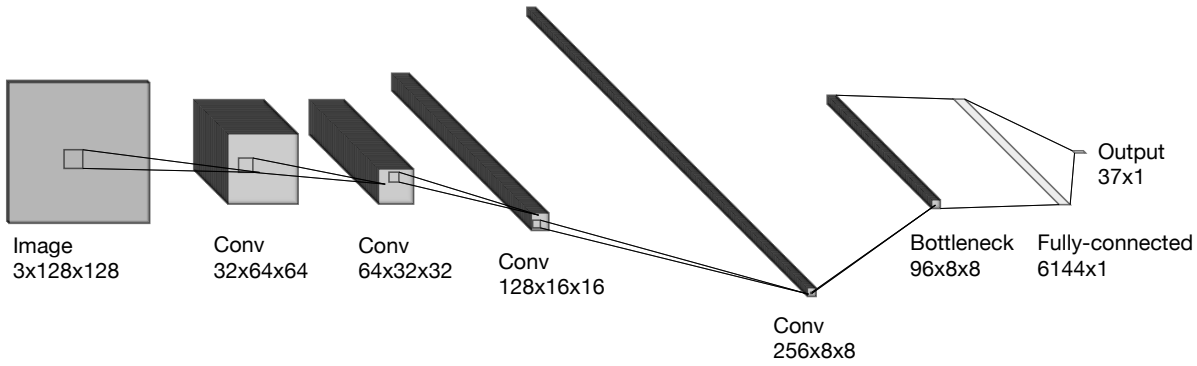

FIG. 4. illustration of the model architecture

The baseline model (figure 4) consists of 4 convolutional blocks, 1 bottleneck block and 1 classification block, with approximately 3 millions trainable parameters. Each convolutional block consists of a convolutional layer, a batch normalization layer, ReLU activation function and a max pooling layer (Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ Max Pooling). Batch normalization layers were used to speed up the training and to provide regularization. The convolutional kernel sizes are 17, 13, 9 and 7 respectively, as we found that larger kernel sizes greatly improved the performance of the model. As our model only have a few layers, using a larger kernel size effectively enlarged the receptive field of a model, which helped it to learn the global features of an image.

The bottleneck block comprises a convolutional layer with kernel size of 1, a batch normalization layer and ReLU activation function (Conv → BatchNorm → ReLU). The purpose of the bottleneck block is to encourage feature extraction by reducing the number of features maps.

For the output, a fully connected layer with a sigmoid layer are used to map the extracted features to the target probabilities. We have also tested Softmax and weighting layers for calculating the weighted probability distributions based on the structure of the decision tree in figure 2, but we found that a simple sigmoid layer performed better in practice.

Results

We trained the baseline model for 240 epochs. The runtime for each epoch is about 80 seconds, and therefore takes 6 hours to complete the training.

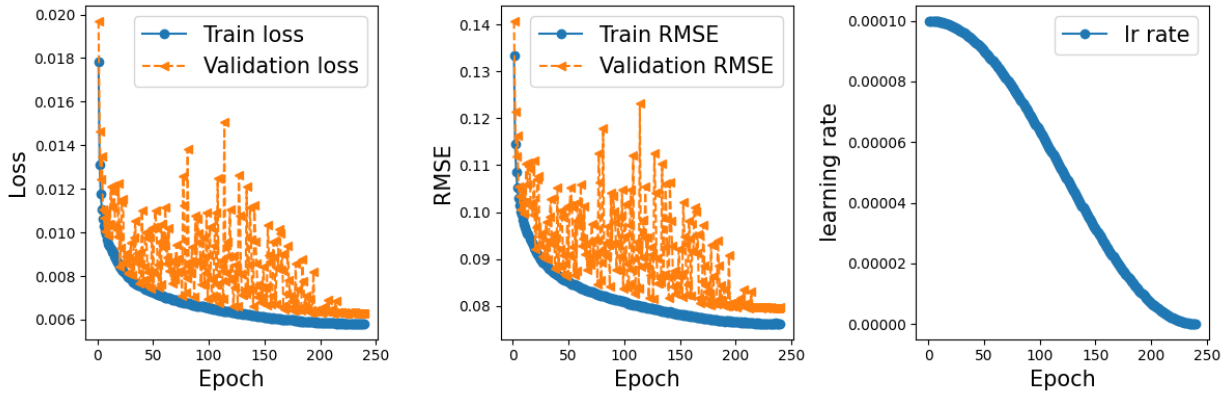

FIG. 5. Training history of loss, RMSE and learning rate

The predictions from the baseline model were submitted to Kaggle and obtained a final score of 0.07742 (cf. winner: 0.07693 [4]). Since the winner also used CNN with a similar approach, it is no surprise that we achieved similar scores.

IMPROVING THE BASELINE MODEL WITH SELF-ATTENTION MODULE

To further improve our baseline model, we included self-attention mechanism, which allows the model to focus on important features. In this section, we give an introduction

to Convolutional Block Attention Module (CBAM), which can be easily plugged into most convolutional neural networks.

Convolutional Block Attention Module (CBAM)

We adopted the Convolutional Block Attention Module (CBAM) proposed by Woo et al. [5]. Unlike the Squeeze-and-Excitation (SE) [6] module that only exploited inter-channel relationships, the authors of CBAM argued that both channel and spatial attentions are important. Therefore, by training the model with a channel attention module (CAM) and a spatial attention module (SAM), the model can learn to apply attention to both channel-wise and spatial-wise features.

Figure 6 and 7 are overview of CBAM from the original paper. The CAM module obtained the channel-wise attention by training a shared multi-layer perceptron (MLP) on the average-pooled and max-pooled single-pixel feature maps. The SAM module obtained the spatial-wise attention simply by training a convolutional layer on the spatial feature map after applying the channel attention and also pooling operations.

Now we can simply insert CBAM into our convolutional blocks:

$$\text{CNN: } (\text{Conv} \rightarrow \text{BatchNorm} \rightarrow \text{ReLU} \rightarrow \text{Pooling})$$

$$\text{CNN + CBAM: } (\text{Conv} \rightarrow \text{BatchNorm} \rightarrow \text{ReLU} \rightarrow \text{CBAM} \rightarrow \text{Pooling})$$

We used a reduction ratio $r = 4$ for the shared MLP in CAM, and a kernel size of 7 for the convolutional layer in SAM.

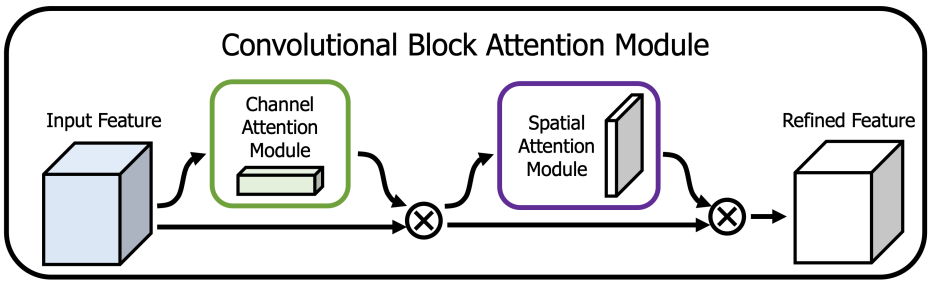

FIG. 6. Overview of the Convolutional Block Attention Module (CBAM) [5]

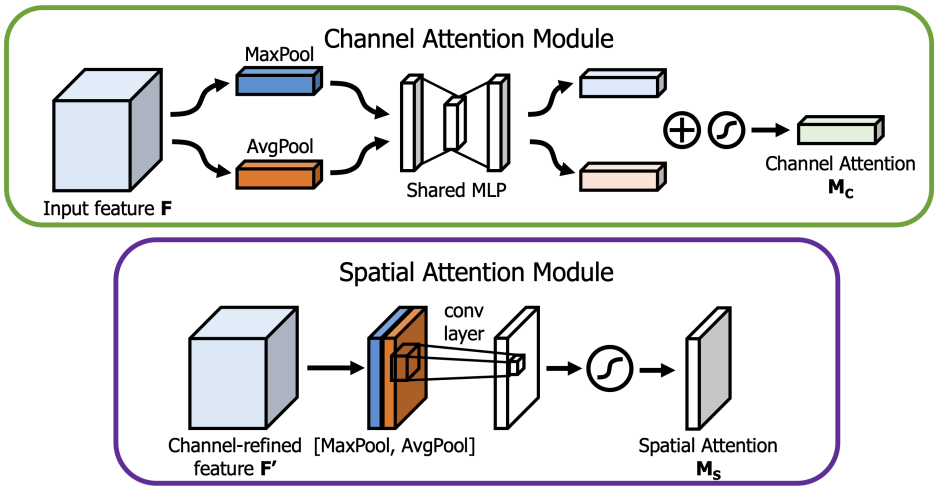

FIG. 7. Overview of the channel attention module (CAM) and spatial attention module (SAM) [5]

Results

The results are presented in table I. Same as the Baseline model, the CBAM model were trained for 240 epochs. Adding the CBAM improved the model score by 0.00049. Although the improvement is small, adding the CBAM did not introduce noticeable overhead, so adding CBAM is favorable in practice.

TABLE I. Score comparison for the Baseline model with self-attention modules

Model	Score	# of Epochs	Total runtime / per epoch (s)
Baseline	0.07742	240	19200 / 80
Baseline + CBAM	0.07693	240	19680 / 82

Using self-attention modules, we successfully reproduced the single model score of 0.07693 of the competition winner [4].

Ablation study

In Figure 8, we have visualized the feature maps in the final convolutional layer using the AblationCam from the `grad-cam` package [7]. The baseline model (middle image) seems to focus on the general central region with a diffused attention. With the CBAM module, the model is able to focus better, as the attention map have same elliptical shape as the galaxy.

However, the attention is not perfect, as it focused on the left side of the galaxy, possibly due to the bright spot nearby.

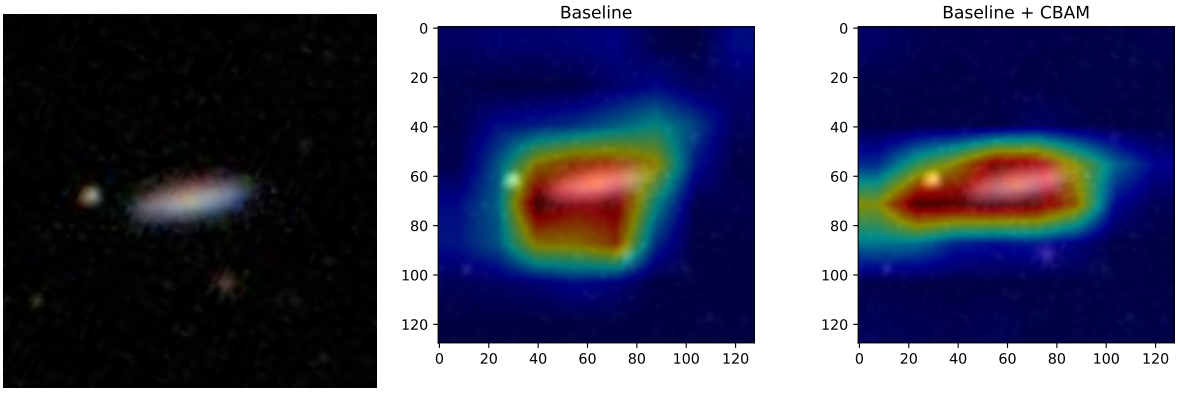

FIG. 8. Feature maps visualization using the AblationCam from the `grad-cam` package [7]

VISION FOUNDATION MODELS

Over the last decade, many work has been done on computer vision models. In this section, we explore the performance of vision foundation models: ResNet, ConvNeXt and Dinov2.

Transfer Learning

Our Galazy Zoo training dataset have only 61578 images, which is relatively small and not enough to train a deep neural network. Luckily, some features from different datasets are actually similar and reuseable. By using a pre-trained model that was trained on a large dataset, and then fine-tuning it on our dataset, we can achieve much better performance and also reduce the training time.

Pre-trained computer vision models are also known as vision foundation models. In this paper, ResNet and ConvNeXt were pre-trained on the ImageNet-1K v1 dataset and Dinov2 was pre-trained on the LVD-142M dataset. These vision foundation models were available in PyTorch as "out of the box" models. To fine-tune it, we simply attach a fully connected layer to the last layer of the model and start training.

Vision Transformer (ViT)

Transformer is a type of neural network that utilizes self-attention mechanism, which have achieved great success in natural language processing (NLP) tasks. In computer classification tasks, ViT split an image into patches and treat them like words in a sequence, as in NLP tasks [8]. They have surpassed the performance of CNN, but they are also more computationally demanding.

ResNet

Training deep neural networks enables the model to learn more complex features. However, the training accuracy was found to degrade rapidly when we blindly stack more layers to a model. To address this issue, He et al. [9] proposed the Residual Network (ResNet) in 2015. The key idea of ResNet is to introduce skip connections that bypass one or more layers, which allows the model to learn the identity function that is easier to optimize than the original function.

ConvNeXt

ConvNeXt is a model proposed by Xie et al. [10] in 2022. Given the success of vision transformers, the authors attempted to push the limits of a pure convolutional network by applying some techniques of vision transformers to a ResNet.

Dinov2

Dino (self-**di**stillation with **no** labels) [11] is a type of self-supervised ViT that uses a teacher-student framework. The teacher is given a global crop of the image, and the students are given a local crop. The students learns to replicate the output of the teacher, while the teacher distill knowledge from the students. In this way, the model can learn to focus on important features of an image, although it is not given any labels. By attaching a fully-connected layer to the model, we can train it to classify the galaxy morphologies. Dinov2 [12] employed various techniques to improve the model, such as using a large curated dataset LVD-142M to pre-train the model.

Results and Discussion

The results are presented in table II. Training a ResNet50 model from scratch (i.e. not pre-trained) for 120 epochs, we achieved a score of 0.07574. Comparing to the baseline model, it achieved a great improvement of 0.00168 with half the number of epochs. Training a pre-trained ResNet50 model for 30 epochs, we achieved a further improvement of 0.00181 to 0.07393, which proves the effectiveness of transfer learning. We also trained a ResNet101 model, which has similar performance to ResNet50.

As for the ConvNeXt Large model, we trained it for 30 epochs and achieved a score of 0.07357. Although it has a slightly better performance than ResNet50, the runtime is significantly longer (1500 seconds per epoch). On the other hand, the Dinov2 Base model achieved the best score of 0.07251 with only 15 epochs, with a moderate runtime of 580 seconds per epoch.

From this result, we conclude that Dinov2 is the best choice among the chosen models, as it achieved the best score with a moderate runtime. However, if computational resources are limited, ResNet50 is a good choice that gives great performance with a shorter runtime.

TABLE II. Score comparison of different models

Model	Score	# of Epochs	Pretrained dataset	Total runtime / per Epoch (s)
Baseline	0.07742	240	/	19200 / 80
Baseline + CBAM	0.07693	240	/	19680 / 82
ResNet50	0.07574	120	/	15000 / 125
ResNet50	0.07393	30	ImageNet-1K v1	3750 / 125
ResNet101	0.07400	30	ImageNet-1K v1	5520 / 184
ConvNeXt Large	0.07357	30	ImageNet-1K v1	45000 / 1500
DINOv2 Base	0.07251	15	LVD-142M	17400 / 580

CONCLUSION

The single model score of the Galaxy Zoo competition winner was successfully reproduced by a Convolutional Neural Network with self-attention module CBAM. In addition, we have compared multiple vision foundation models, with Dinov2 achieving the best score of 0.07251 with a moderate runtime, while ResNet50 having a great performance of 0.07393 with a shorter runtime. The effectiveness of transfer learning was also demonstrated.

In this paper, we have successfully demonstrated the use of machine learning to automate the classification of galaxy morphologies, by exploiting the invariances of galaxy images and the ability of deep neural networks.

[1] C. Lintott, K. Schawinski, S. Bamford, A. Slosar, K. Land, D. Thomas, E. Edmondson, K. Masters, R. C. Nichol, M. J. Raddick, A. Szalay, D. Andreescu, P. Murray, and J. Vandenberg, Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies*, Monthly Notices of the Royal Astronomical Society **410**, 166 (2010), https://academic.oup.com/mnras/article-pdf/410/1/166/18442057/mnras0410-0166.pdf.

[2] K. W. Willett, C. J. Lintott, S. P. Bamford, K. L. Masters, B. D. Simmons, K. R. V. Casteels, E. M. Edmondson, L. F. Fortson, S. Kaviraj, W. C. Keel, T. Melvin, R. C. Nichol, M. J. Raddick, K. Schawinski, R. J. Simpson, R. A. Skibba, A. M. Smith, and D. Thomas, Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey, Monthly Notices of the Royal Astronomical Society **435**, 2835 (2013), https://academic.oup.com/mnras/article-pdf/435/4/2835/3372631/stt1458.pdf.

[3] AstroDave, AstroTom, C. R. . Winton, joycenv, and K. Willett, Galaxy zoo - the galaxy challenge, `https://kaggle.com/competitions/galaxy-zoo-the-galaxy-challenge` (2013), kaggle.

[4] S. Dieleman, My solution for the galaxy zoo challenge, GitHub repository (2014).

[5] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, Cbam: Convolutional block attention module, in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018).

[6] J. Hu, L. Shen, and G. Sun, Squeeze-and-excitation networks, in *2018 IEEE/CVF Conference*

on Computer Vision and Pattern Recognition (2018) pp. 7132–7141.

[7] J. Gildenblat and contributors, Pytorch library for cam methods, `https://github.com/jacobgil/pytorch-grad-cam` (2021).

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale (2021), arXiv:2010.11929 [cs.CV].

[9] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition (2015), arXiv:1512.03385 [cs.CV].

[10] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, A convnet for the 2020s (2022), arXiv:2201.03545 [cs.CV].

[11] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, Emerging properties in self-supervised vision transformers (2021), arXiv:2104.14294 [cs.CV].

[12] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, Dinov2: Learning robust visual features without supervision (2024), arXiv:2304.07193 [cs.CV].