Galaxy Image Classification

Individual Report

Author: Mishkin Khunger

DATS6203: Machine Learning II

The George Washington University

May 1, 2020

Table of Contents

Introduction ---------- 3

Description of Individual Work ---------- 3

Portion of the work I did ---------- 3

- Transfer Learning and Pretrained Networks ----------3
- ResNet ----------4
- ResNet50 ---------- 5
- Predict Function ---------- 6

Results ---------- 8

Conclusion ---------- 9

References ---------- 10

Introduction

Galaxy Zoo - Classify the morphologies of distant galaxies in our Universe. Dataset asks to analyze the JPG images of galaxies and find automated metrics that reproduce the probability distributions derived from human classifications. For each galaxy, the user has to determine the probability that it belongs in a particular class. This project aims to develop a Neural Network to classify Galaxy Images for 37 different categories with probabilities ranging between 0 to 1 which makes it a multi-label classification problem.

Description of Individual Work

After using a 2-layered CNN model with and without augmentation, we tried adding layers to our network, changing the optimizers, trying higher number of epochs, playing with the batch size and learning rates but we could not reach a better performing network. I then decided to use transfer learning to improve our modelling. I read about a lot of pre-trained models. I particularly chose ResNet because of it's exclusive property of ability to skip one or more layers.

Portion of the work I did

Transfer Learning and Pretrained Networks

Transfer learning is the process where learning of a new task depends on the previous learned tasks. Generally, to train a deep neural network we require large amounts of dataset, using a pre-trained network solves this problem to a great extent. We can achieve better accuracy with less amount of data using a pre-trained network as the network utilizes knowledge from previously learned tasks and applies them to newer, related ones.

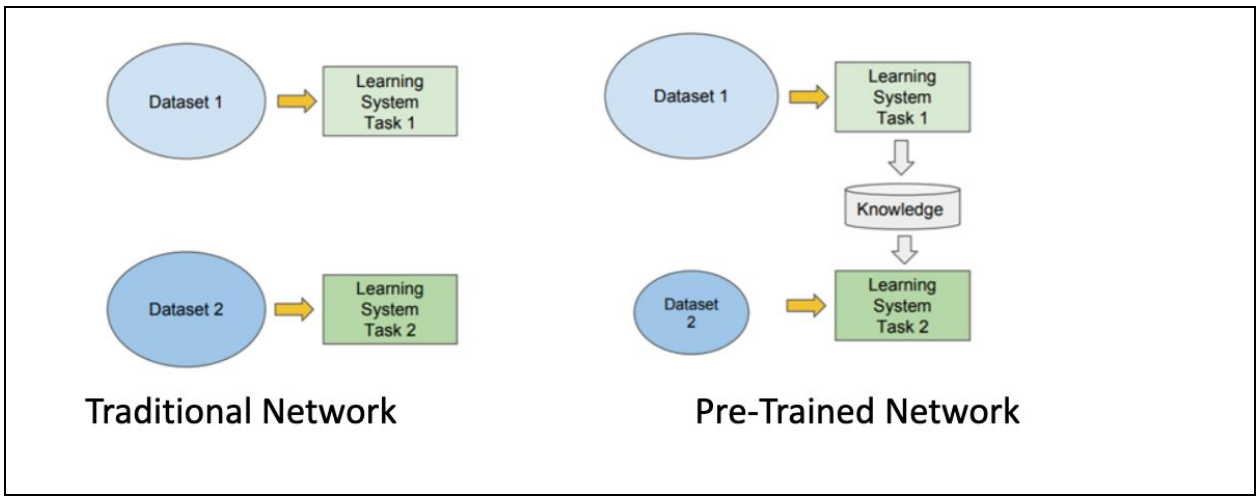

Figure 1. Difference between traditional and pre-trained network

ResNet

The core idea of ResNet is introducing an "identity shortcut connection" that skips one or more layers. ResNet stands for Residual Neural Network. It is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of the ImageNet challenge in 2015. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients. There are various variants of ResNet network with different number of layers(Figure.2)-

1.) Resnet18 - 18 layers
2.) Resnet34 - 34 layers
3.) Resnet50 -50 layers
4.) Resnet101 -101 layers
5.) Resnet152 - 152 layers

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2. ResNet Architecture

Source: https://pytorch.org/hub/pytorch_vision_resnet/

I tried implementing resnet18 and resnet34 but the train and test loss scores were highly inappropriate. The RMSE graph obtained was not at all smooth. I then decided to go ahead with resnet50.

ResNet50

ResNet50 contains 50 layers where there are 4 convolutional layers, the 1st convolutional layer consists of 3 Bottleneck blocks, the 2nd convolutional layer consists of 4 Bottleneck blocks, the 3rd convolutional layer consists of 6 Bottleneck blocks and the 4th convolutional layer consists of 3 Bottleneck blocks. ResNet50 is originally trained on ImageNet dataset where there are 1000 classes, and since there are 37 classes in our dataset, we had to modify the output of the network by adding a linear layer with 37 output classes. (Figure.3)

```
(layer4): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=2048, out_features=37, bias=True)
```

Figure 3. Modified Output Layer of ResNet

I also worked on creating the predict function-

Predict Function

Our test dataset contained 79975 test images. I transformed the images by cropping and resizing them the same as our training. I then called the .pt model which we trained using the 2-layered CNN model with data augmentation, as it

was our best model. I used sigmoid as the predict function as we wanted to evaluate probability distribution of the classes.

I converted our output into a csv format. Figure.4 shows the probability distribution of the 37 classes in csv format.

GalaxyID	Class1.1	Class1.2	Class1.3	Class2.1	Class2.2	Class3.1
100018	0.000121593235	0.0001043031	0.00014623384	0.00015801816	0.00010088087	0.00011591868
100037	0.00016176129	0.00012848586	0.00018308878	0.00018152053	0.00012723106	0.00014500265
100042	0.00013406927	9.720334E-05	0.00014668492	0.0001322939	8.328794E-05	0.00010947621
100052	0.00011152039	0.00012282172	0.00012160878	0.000119783705	8.8597866E-05	0.0001156548
100056	0.00015400231	9.9473684E-05	0.0001502225	0.00014657559	0.00010210796	0.00011599785
100058	0.0001717886	0.00016598876	0.00015994594	0.00017102616	0.00012491681	0.00016738535
100062	0.0001519452	0.00015299735	0.00021583414	0.00016630895	0.0001594071	0.00012289434
100065	0.00013765975	9.77332E-05	0.00013675948	0.00014096318	8.947015E-05	0.000119656645
100071	0.00014234564	0.000111103574	0.00012972431	0.00012904833	9.5174015E-05	0.00011195352
100076	0.0001497537	0.00011036927	0.00014063787	0.00010188124	0.00013292844	0.00015450739
100084	0.00013615067	0.00012452103	0.00013401649	0.00014509946	9.7673204E-05	0.0001212614
100085	0.00010485218	0.00010085816	0.00012892374	0.00013205949	8.536978E-05	0.00010340676
100094	0.00013348123	9.9437355E-05	0.00011238361	0.00011823265	0.00010427794	9.377853E-05
100104	0.00012116408	0.00011295787	0.00014525064	0.000145001	9.20053E-05	0.00011202784
100171	0.00012564313	0.00012155754	0.00012435645	0.00011073529	8.193456E-05	9.307668E-05
100183	9.913779E-05	8.15442E-05	0.0001265297	0.000117096075	0.0001038444	9.0667934E-05
100186	0.00013136257	0.0001053985	0.00014613167	0.00014593116	9.165345E-05	0.00012859727
100190	0.00015874516	0.00011476209	0.0001546787	0.00014657225	0.00010745157	0.00012650389
100208	0.00012686169	0.00010338271	0.00015484574	0.00014247859	0.00010179831	0.0001316053
100222	0.00016336981	0.00012750397	0.00019386385	0.00013155436	0.0001052303	0.00013531031

Figure 4. csv output for testing set

Results

I tried two types of optimizers- SGD and Adam, Adam gave me a comparatively better result. I started with a learning rate of 0.1, then I kept on decreasing it till I got reasonable loss values. I tried different batch sizes-64,256,512,1024 but I got "Cuda out of memory issue" in most of them.

On training the network on our dataset with 150 epochs, learning rate as 0.000001 and Adam as the optimiser and batch size of 256, I obtained a graph using the training and validation losses. It can be observed from figure.3 that the training and the validation losses obtained using ResNet and our 2-layered CNN model are similar. Hence, to further get a deeper insight we plotted a RMSE curve.

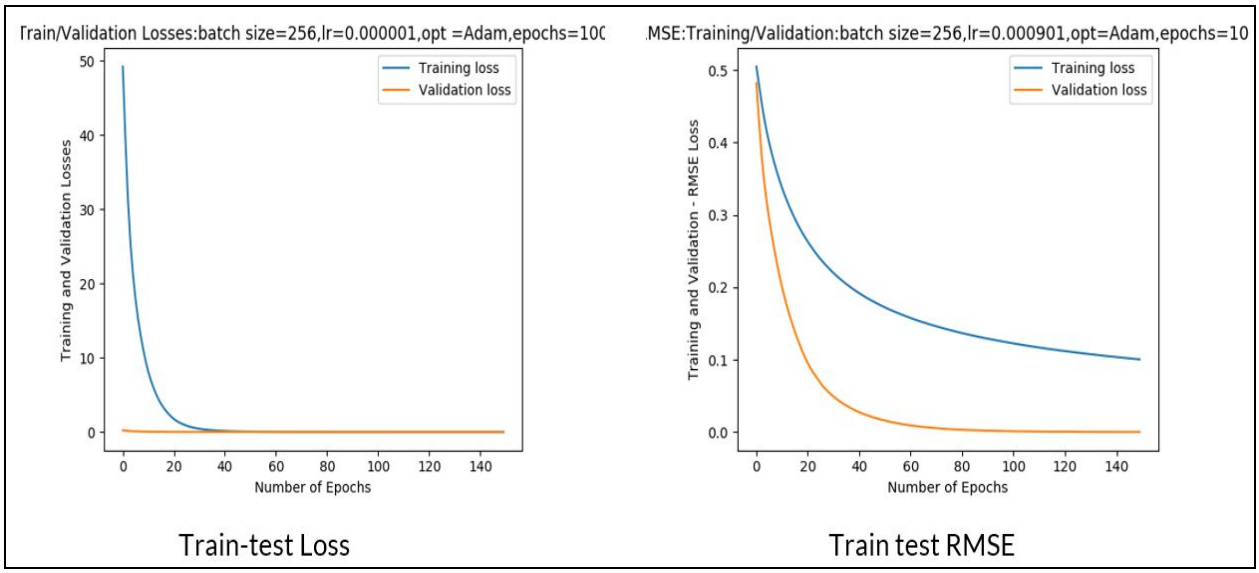

Figure 5. Train/Validation RMSE curve using Resnet

The RMSE curve clearly shows a significant lag between the train and the test RMSE. Hence, I feel it is not a good fit for our prediction.(Figure.5)

Summary and conclusions

The purpose of this study was to classify the galaxy images for 37 different labels, which made it a multi-label classification problem. Training parameters like batch size, learning rate, and optimizer were checked with different values for the analysis of training and validation losses. And, it was finally concluded that the model was only learning on specific values of training parameters: batch-size = 1024, lr = 0.001, Optimizer = Adam. The CNN network was also tested with more than 2 layers however, again the losses and RMSE on training and validation set were very high. Therefore, only the 2layer basic conv2d layer network was utilized.

Pre-trained networks do not perform well on all types of datasets, ResNet does provide a wide variety of models with different number of layers and complexities, one can achieve a good performance by experimenting with skipping some layers.
Changing the learning rate has a great impact on the train and validation losses. Changing the batch size can help rectify the memory constraints(Cuda out of memory errors.)

I learned a lot of things from this project:

- Working with a multi-label classification problem.
- Designing a CNN architecture and how adding and removing a single layer creates a big impact.
- Importance of data augmentation as it helps in capturing the complex features of the image.
- Using a pre-trained model on your own dataset just by modifying it according to your requirements.
- Importance of RMSE score to evaluate your model.

Code Reference:
For my individual work, I referred around 25% from the internet.

References

- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

- https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

- https://www.kaggle.com/helmehelmuto/keras-cnn

- https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

- https://medium.com/@anishbatra95/winning-kaggles-galaxy-zoo-challenge-in-2019-resnet-xception-1767a786946e

- https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/

- https://github.com/benanne/kaggle-galaxies