Galaxy Image Classification

(Group 1)

Authors: Jyoti Sharma, Mishkin Khunger, Tanvi Hindwan

DATS6203: Machine Learning II

The George Washington University

May 1, 2020

Table of Contents

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.

Dataset Details

The Galaxy Zoo dataset used for this research is obtained from Kaggle. It comprises 61,578 galaxy images in the training set, 79,975 in the testing set and a csv file with probability distributions for the classifications for each of the training images. The data set has 37 labels representing the morphology (or shape) of the galaxy in 37 different categories. These morphologies are related to probabilities for each category; a high number (close to 1) indicates that many users identified this morphology category for the galaxy with a high level of confidence. Low numbers for a category (close to 0) indicate the feature is likely not present.

Network Architecture

This section puts forward the deep learning network architecture developed for training the algorithm used to analyze the problem statement.

Convolution Neural Network is a deep learning algorithm which takes parameters like input images, kernel size, strides and feature maps etc. to develop an architecture over which we train our model to learn and classify the problem statement. In the project, the images were resized and reshaped so as to make them manageable by the network during preprocessing steps. The network was built using 2 convolution layers with batch normalization and maxpooling (Figure 1).

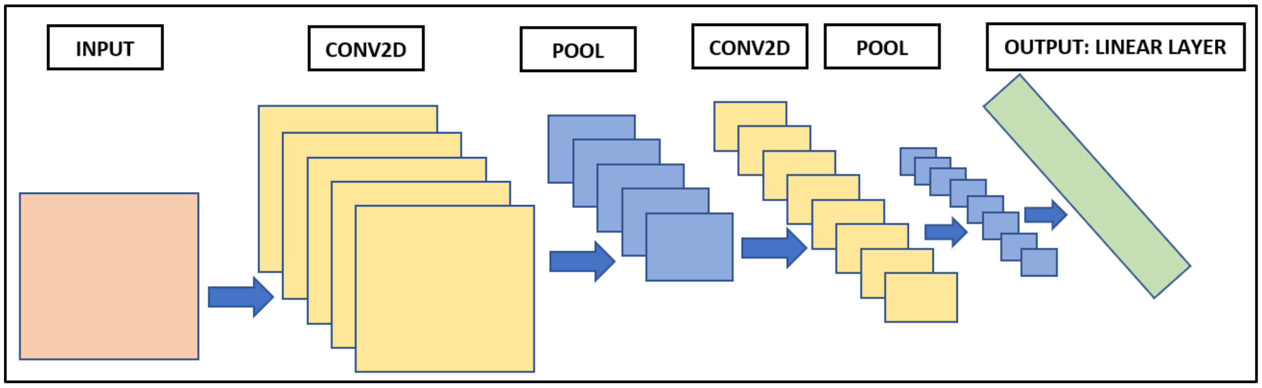
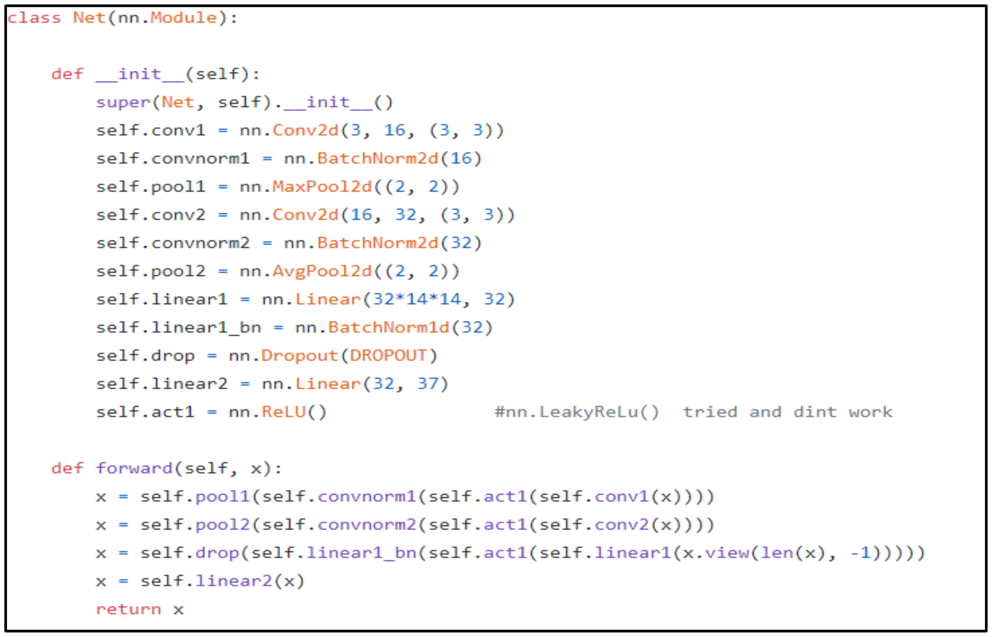


Figure 1. Network architecture

The flow of algorithm implementation in building the network layers for training can be seen in figure 2. Layer 1 is conv2d with 3 channels (RGB), input shape 64x64, feature maps 16 and kernel size 3 which is followed by batch normalization layer and maxpool layer. As we know, Batch normalization not only increases the speed of training it also reduces the sensitivities to initial weights and at the same time maxpool layer was also applied to extract sharp, important and smooth features out of the image. Layer 2 also followed the same pattern of flow with feature maps 32, kernel size 3 followed by batchnorm and average pooling to output a fully connected linear layer with 37 classes.



```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, (3, 3))
        self.convnorm1 = nn.BatchNorm2d(16)
        self.pool1 = nn.MaxPool2d((2, 2))
        self.conv2 = nn.Conv2d(16, 32, (3, 3))
        self.convnorm2 = nn.BatchNorm2d(32)
        self.pool2 = nn.AvgPool2d((2, 2))
        self.linear1 = nn.Linear(32*14*14, 32)
        self.linear1_bn = nn.BatchNorm1d(32)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(32, 37)
        self.act1 = nn.ReLU()                #nn.LeakyReLu()  tried and dint work

    def forward(self, x):
        x = self.pool1(self.convnorm1(self.act1(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.act1(self.conv2(x))))
        x = self.drop(self.linear1_bn(self.act1(self.linear1(x.view(len(x), -1)))))
        x = self.linear2(x)
        return x
```

Figure 2. CNN Network Flow

Experimental Setup

This section puts forwards the research methods that have been applied to classify galaxy images.

Data Preprocessing

The original size of the images in the dataset was 424x424 along with 37 weighted probabilities that have to be predicted for each image in the testing set. The images were resized and standardized in the training set while retaining color for the images. This process only cuts out the noise in the outer part of the galaxy's images. For almost all the images the main data was only in the center of the images, so we center cropped the whole image using the resize function (Figure 3) and reduced the size of the image to 256 x 256. The final image shape used for the training network was set to (3 x 64 x 64).

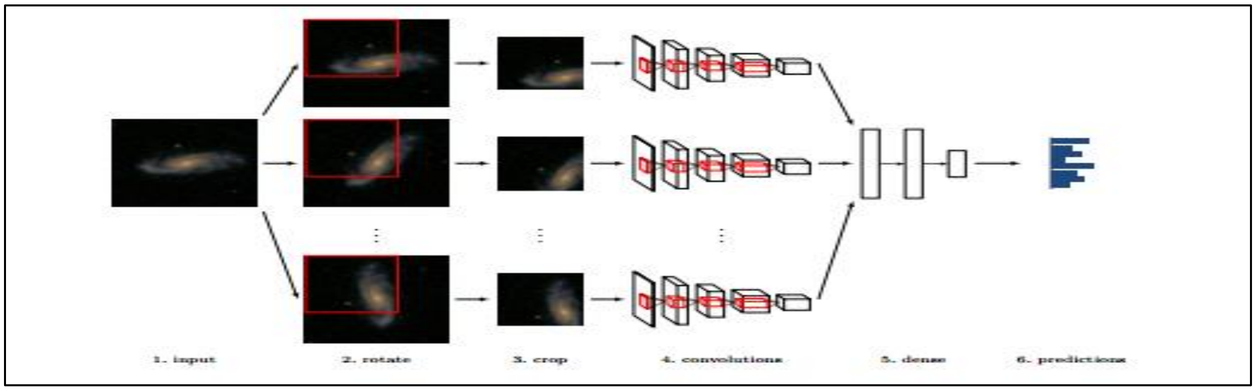
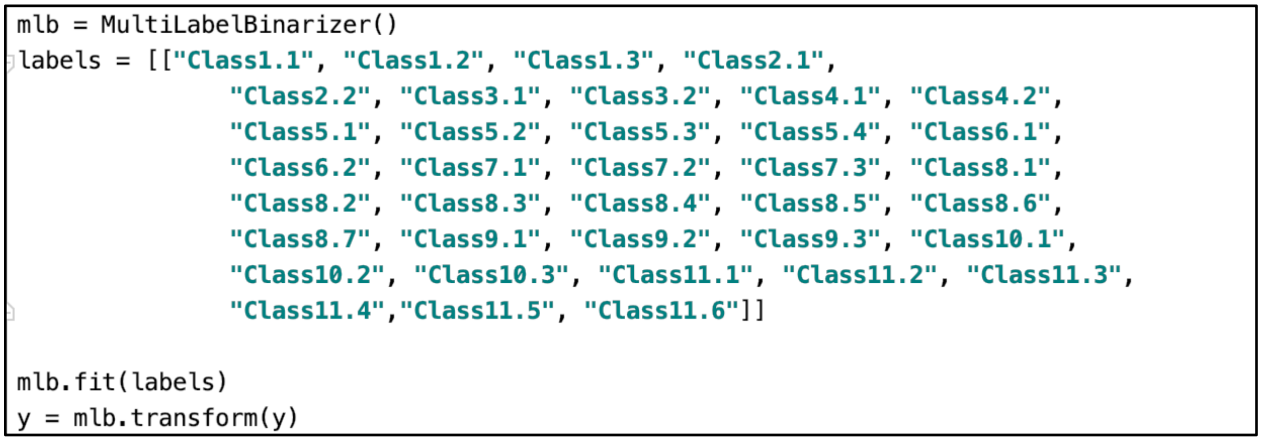


Figure 3. Image Resize and Reshape https://www.technologyreview.com/2015/04/03/11482/how-machine-vision-is-reinventing-the-study-of-galaxies/

The target were multi-labels with 37 different classes so they were One-hot encoded using Multi-label binarizer (Figure 4) from the sklearn.preprocessing library.



```
mlb = MultiLabelBinarizer()
labels = [["Class1.1", "Class1.2", "Class1.3", "Class2.1",
                "Class2.2", "Class3.1", "Class3.2", "Class4.1", "Class4.2",
                "Class5.1", "Class5.2", "Class5.3", "Class5.4", "Class6.1",
                "Class6.2", "Class7.1", "Class7.2", "Class7.3", "Class8.1",
                "Class8.2", "Class8.3", "Class8.4", "Class8.5", "Class8.6",
                "Class8.7", "Class9.1", "Class9.2", "Class9.3", "Class10.1",
                "Class10.2", "Class10.3", "Class11.1", "Class11.2", "Class11.3",
                "Class11.4","Class11.5", "Class11.6"]]

mlb.fit(labels)
y = mlb.transform(y)
```

Figure 4. Multi-Label Binarizer

Further the data was split into training and validation set for modeling with respect to torch format to feed into CNN architecture.

Modeling:

To classify the galaxy images for 37 different classes with their probabilities, the CNN network architecture was trained using PyTorch as the framework. The model was trained using MSE as the performance index with Adam as optimizer. As the problem statement required to find the probabilities ranging from 0 to 1 therefore Sigmoid function was applied at the output logit of our model.

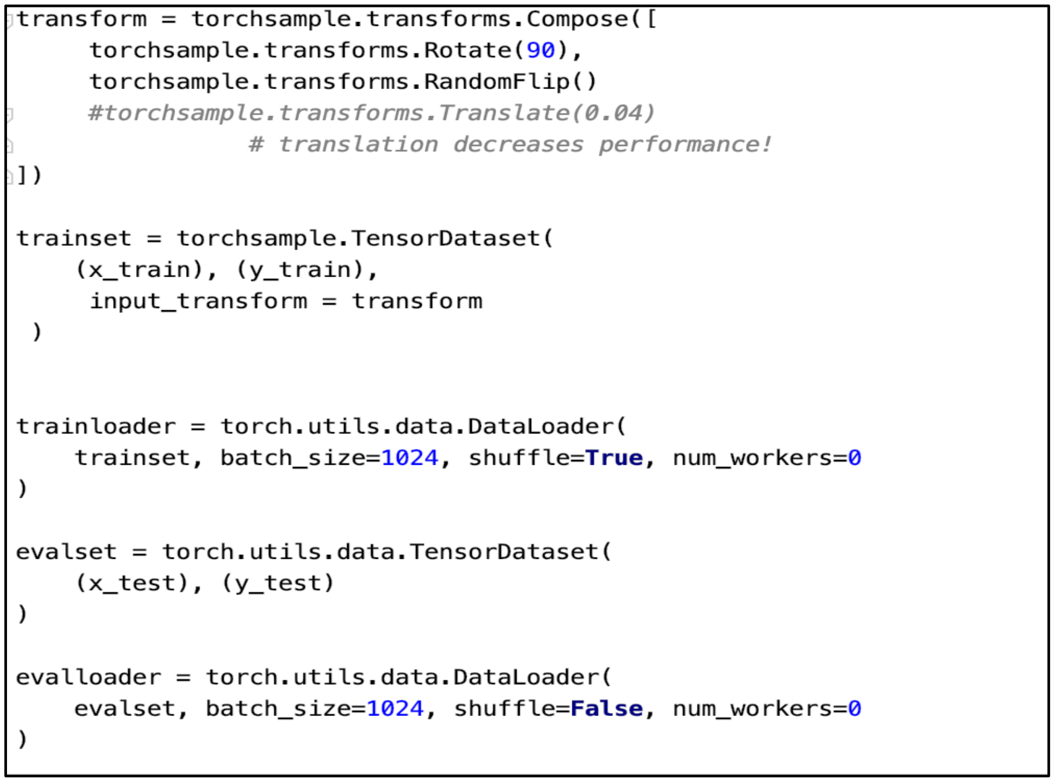
RMSE was used as the evaluation metrics for analysis. Various batch sizes were tried and experimented like 256, 512, 128, 32, 64, 1024 with different learning rates like 0.01, 0.001, 0.0001, 0.5. These training parameters like batch sizes and learning rates were experimented to analyze the losses over the epochs of training and validation sets.

Data Augmentation:

To prevent the dataset from overfitting and to get better RMSE results over training and validation sets data augmentation was performed**.** This technique was implemented on the torch tensors using the 'torchsample' package which is a data loader utility/module in PyTorch for image transformation.

To transform the images, two types of augmentation (Figure 5) functionality was used:

- Rotation: rotating the images with angle between 0° and 90°
- Random Flip: flip the given image randomly with a given probability



```
transform = torchsample.transforms.Compose([
     torchsample.transforms.Rotate(90),
     torchsample.transforms.RandomFlip()
     #torchsample.transforms.Translate(0.04)
                # translation decreases performance!
])

trainset = torchsample.TensorDataset(
    (x_train), (y_train),
     input_transform = transform
 )


trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=1024, shuffle=True, num_workers=0
)

evalset = torch.utils.data.TensorDataset(
    (x_test), (y_test)
)

evalloader = torch.utils.data.DataLoader(
    evalset, batch_size=1024, shuffle=False, num_workers=0
)
```

Figure 5. Data Augmentation

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.

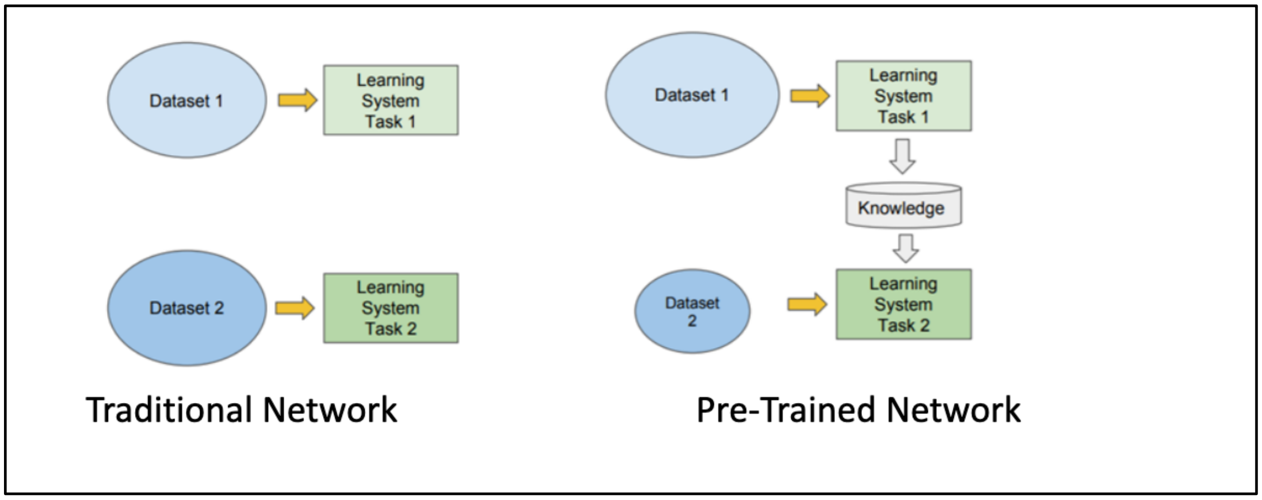
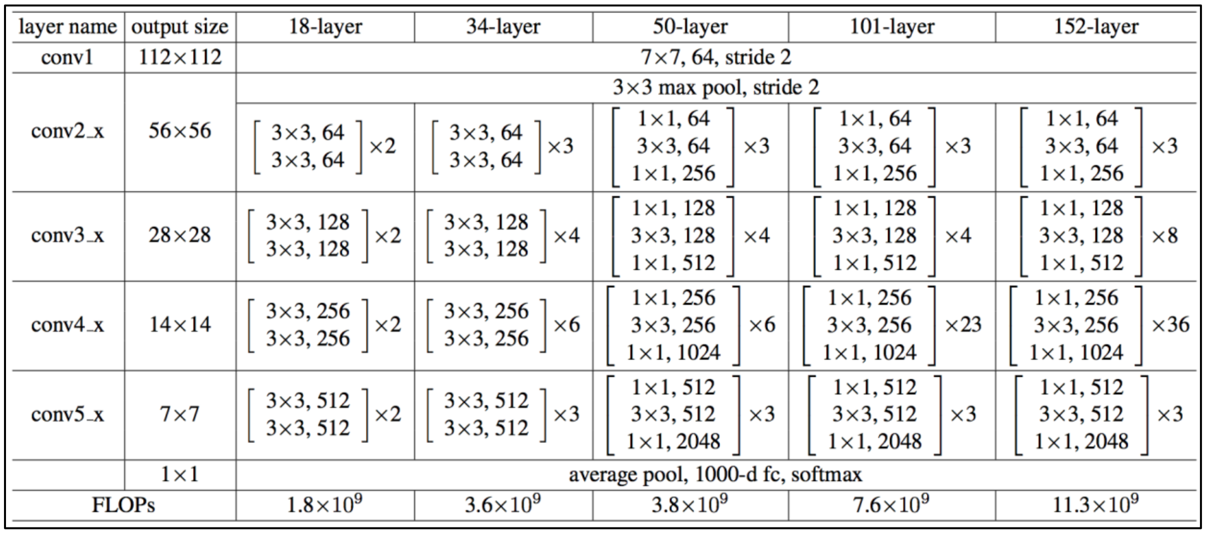


Figure 6. Difference between traditional and pre-trained network

ResNet

ResNet stands for Residual Neural Network and the main idea behind ResNet is introducing an “identity shortcut connection” that skips one or more 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

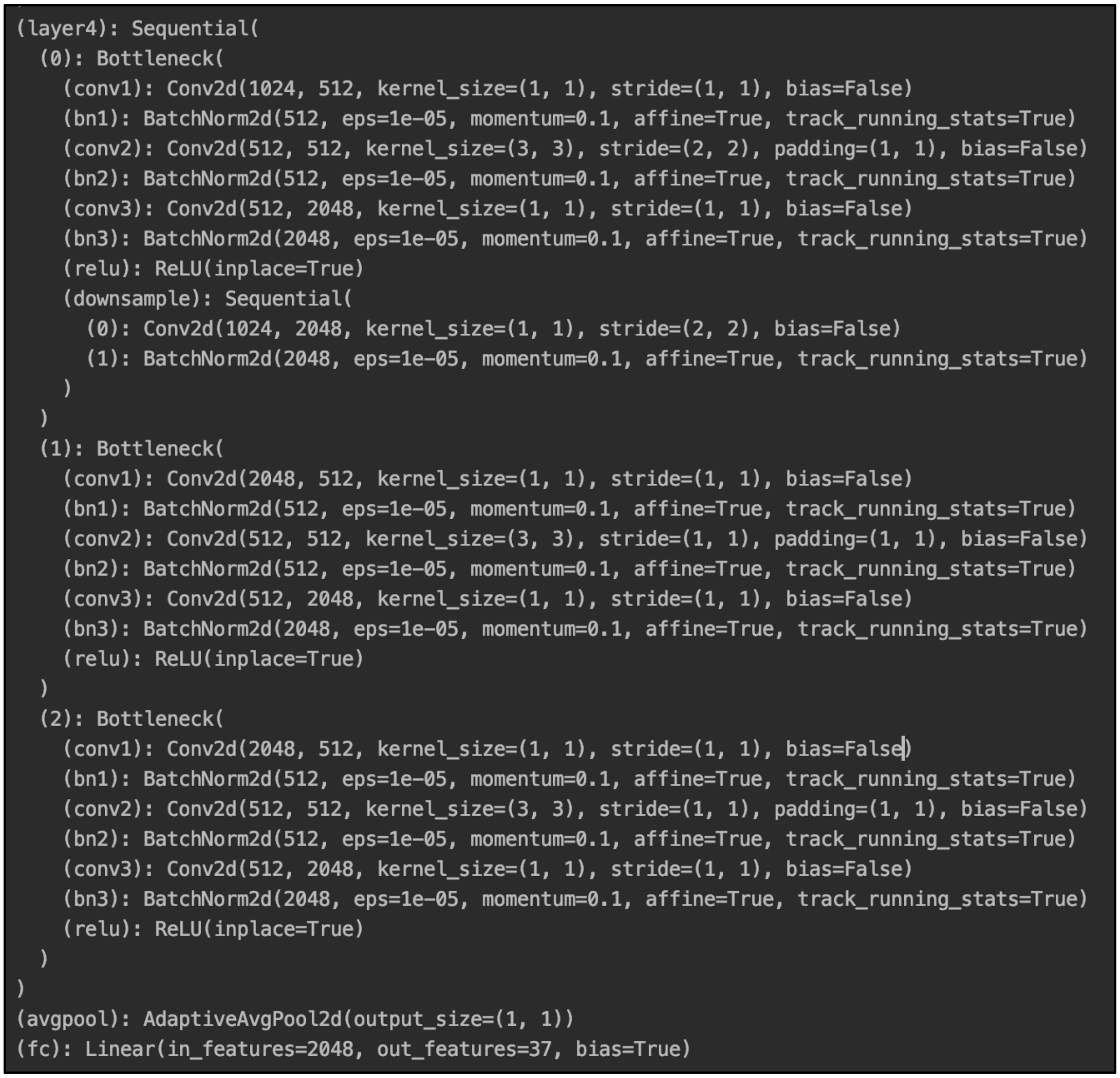
Figure 7. ResNet Architecture

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

ResNet50

ResNet50 contains 50 layers (Figure 7) 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. There are over 24 million parameters. ResNet50 was originally trained on ImageNet dataset where there were 1000 classes, and since there were 37 classes in our dataset, the output layer of the network (Figure8) was modified by adding a linear layer with 37 output classes.



```
(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 8. Modified Output Layer of ResNet corresponding to our dataset

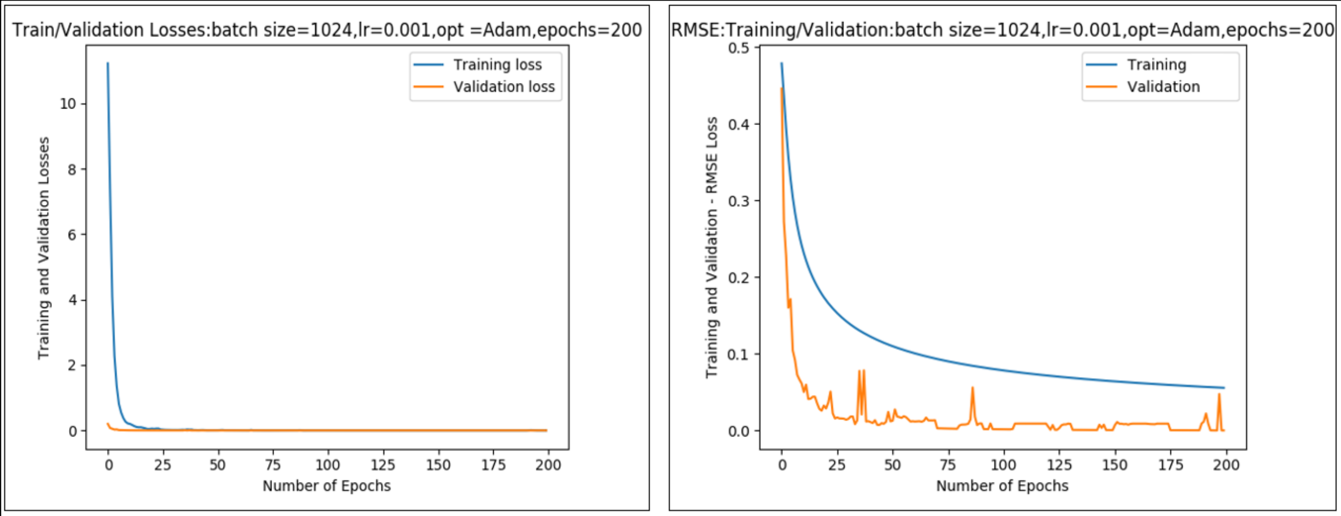
Predict Function:

The dataset had 79,975 test images. So, the predict function was created to check the results of an unseen dataset. The testing set was classified using CNN trained model with sigmoid function at the output layer to get the results as probabilities ranging between 0 to 1.

Results

In this section, the results that were obtained from the research methods are examined for Galaxy Image Classification.

Initially the model was trained over the basic CNN 2-layer conv2d network and analyzed that the training and validation losses are very apart from each other. Therefore, the RMSE for each epoch cycle was analyzed for the same with 200 epochs, batch size 1024 and learning rate 0.001 and estimated that RMSE at each epoch (Figure 9) is giving results with a lag between training and testing which should not be there.



Train/Validate Set Losses

RMSE on Train/validate Set

Figure 9. Training and testing losses and RMSE

Therefore, data augmentation was applied with the same hyperparameters and it was analyzed that the training and validation losses improved with small differences. The RMSE curve also got smoother and decreased with the same rate as that of the training set as shown in figure 10.

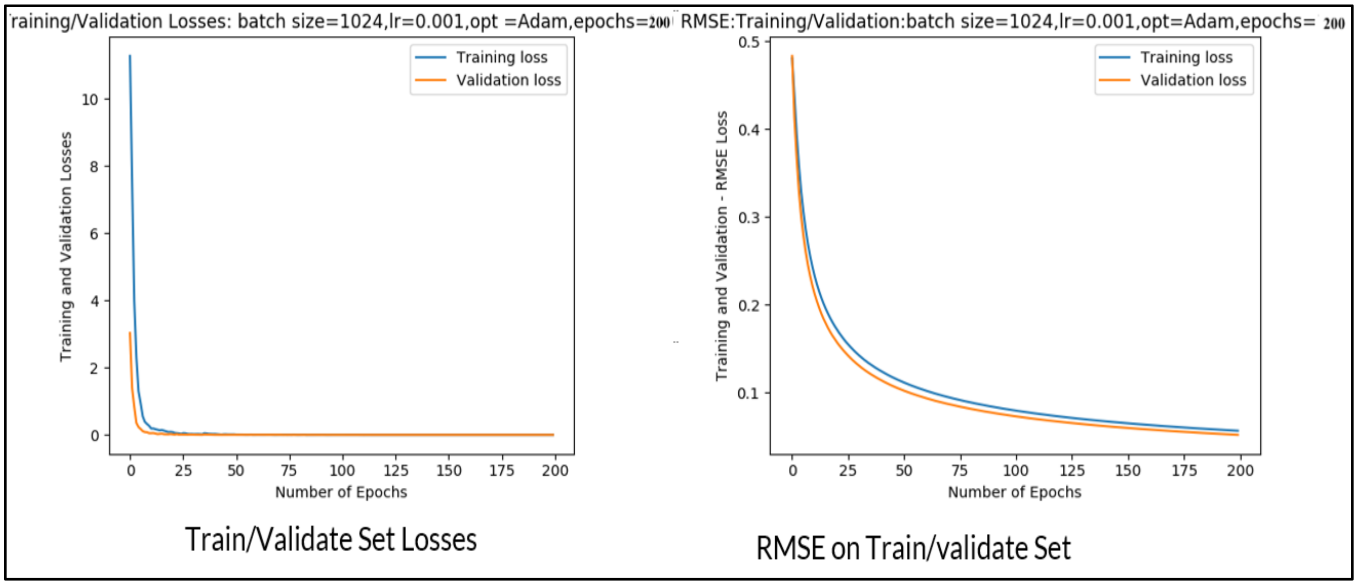


Figure 10. Training and testing losses and RMSE (Data Augmentation)

For making more accurate predictions for the final model, some pretrained network models were also tried among which ResNet50 was successfully utilized with galaxy zoo dataset. On training the ResNet50 with 150 epochs, learning rate as 0.000001, Adam as the optimizer, and batch size of 256 (Figure 11) the training and validation loss graph was obtained which was quite similar to that of the custom model developed earlier. ResNet18 and ResNet34 were also tried, but results were not favorable.

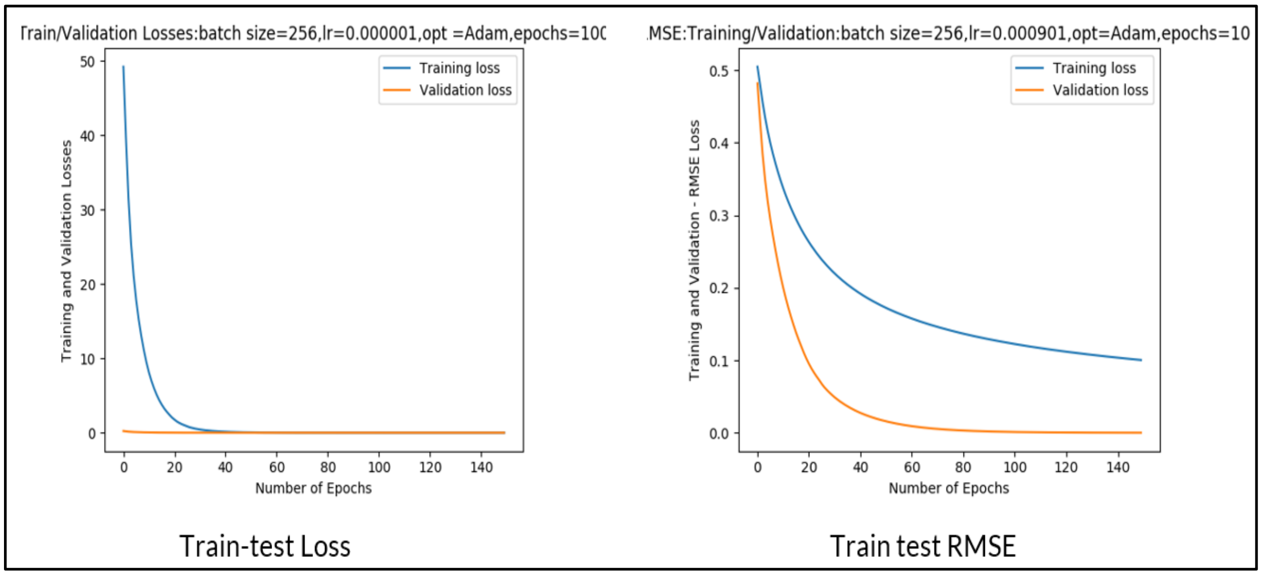
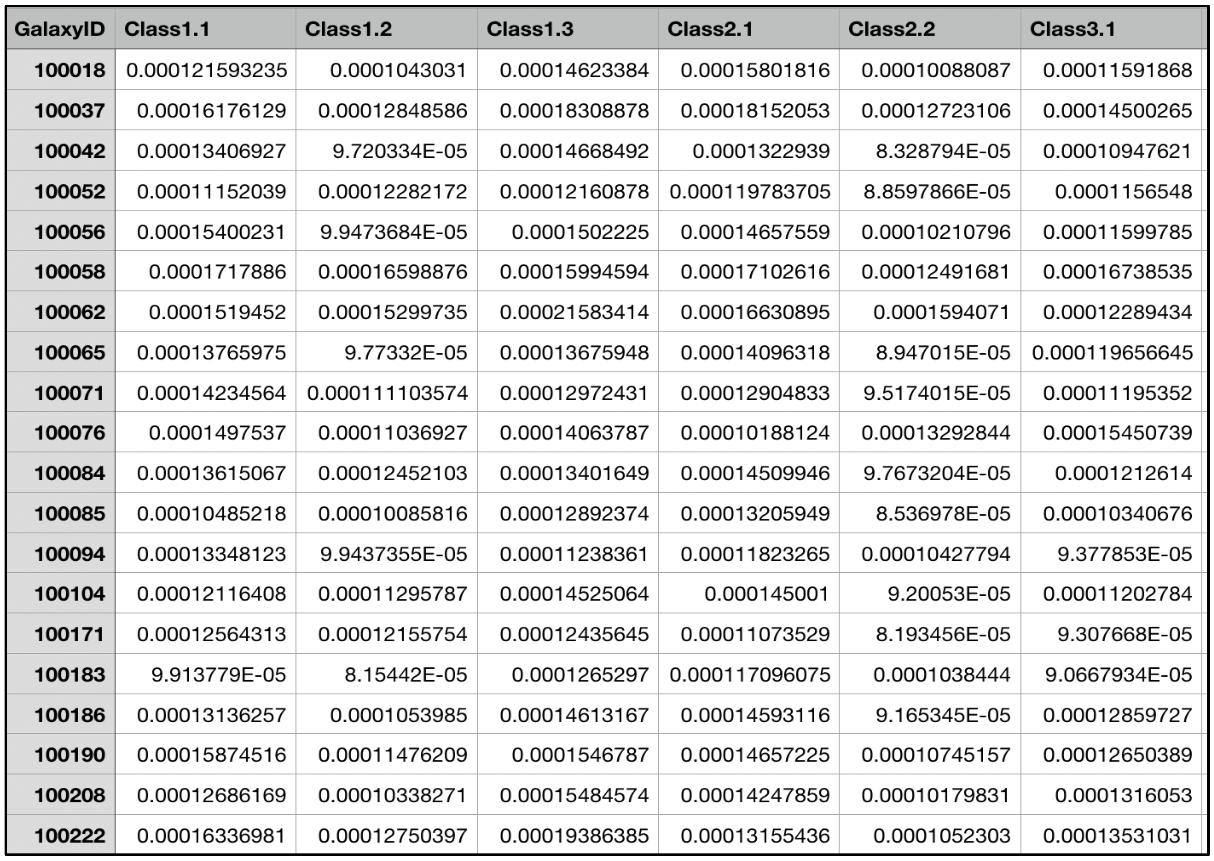


Figure 11. Train/Validation loss and RMSE (Resnet Model)

However, to get a deeper insight RMSE curve was also plotted for each epoch which also gave the significant lag between the train and the test RMSE which was not good.

So, after comparing all three models, we found CNN model with data augmentation as our best fit model. And then the test images were predicted for 37 different classes over that. Figure 12 shows the output in csv format obtained with probabilities for each class.



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 12. Example of the output for testing set

Conclusion

In this section, research is concluded, and key results are provided.

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.

Also, on comparing the two CNN models (with and without augmentation) it was found that the RMSE curve got smoother with data augmentation which determined it as the best fit model. Even ResNet50 (pretrained model) was tried but RMSE values were significantly apart for training, validation, and testing set. Therefore, a data-augmented CNN model was accepted for making predictions on a testing set (unseen images) with RMSE score of 0.22. For future work, more pretrained models can be applied and tested (except ResNet18 and Resnet34). The training and validation losses can be further improved using more data augmentation techniques to capture more complex features of the images.

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://github.com/amir-jafari/Deep-Learning

- 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