Galaxy Image Classification
(Individual Report)

Author: Tanvi Hindwan

DATS6203: Machine Learning II

The George Washington University

May 1, 2020

Table of Contents

Introduction

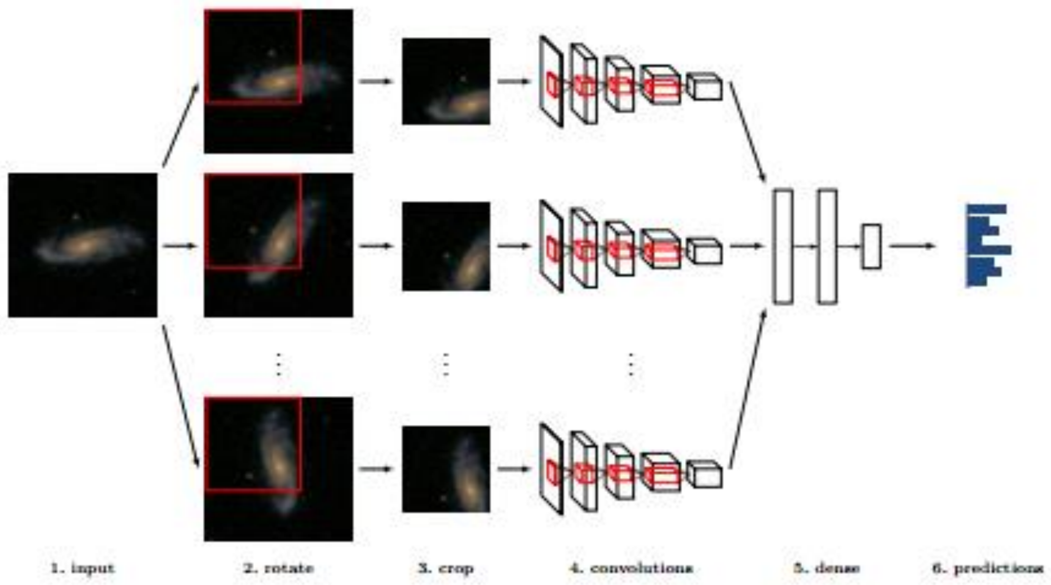
For this project my teammates Jyoti and Mishkin and I collaborated to develop a Neural Network which classify Galaxy Images for 37 different categories with probabilities ranging between 0 to 1. It is a multi-label classification problem and the dataset used for this research is Galaxy Zoo Data set which is sourced from Kaggle.

Description of individual work.

I mostly worked in the data preprocessing and applying data augmentation to the CNN model part in the project.

Data Preprocessing

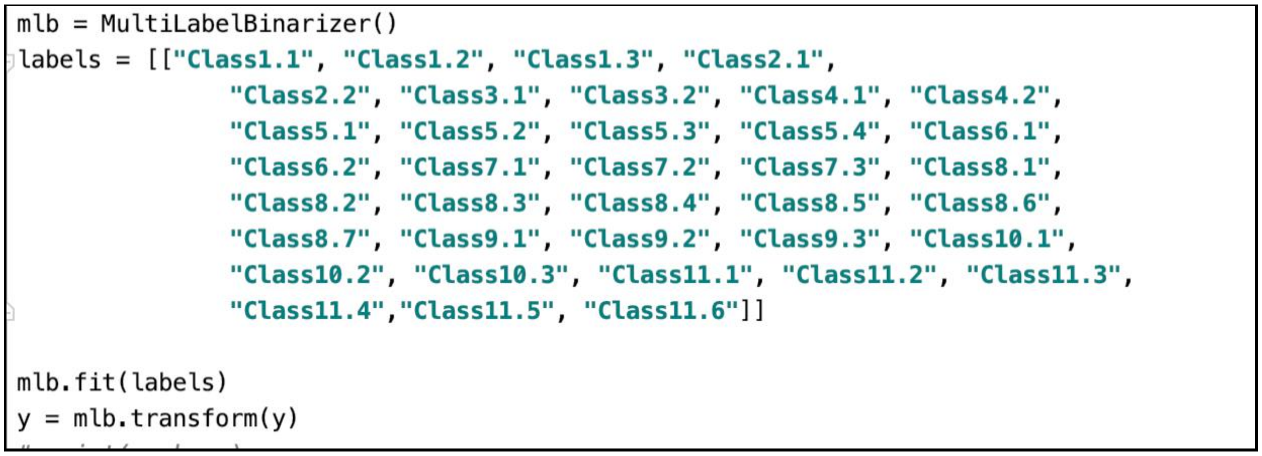
The dataset comprises of 61,578 images of the galaxies in the training set and 79,975 in the testing set. The original size of the images in the dataset was 424x424 along with 37 weighted probabilities that have to be predicted for each image in the testing set. The images were resized and standardized 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 I center cropped the whole image using the resize function 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 1 shows how the images are resize and re-shaped for data modelling



https://www.technologyreview.com/2015/04/03/11482/how-machine-vision-is-reinventing-the-study-of-galaxies/

Figure 1. Data Preprocessing

The data set has 37 labels representing the morphology (or shape) of the galaxy in 37 different categories. The target were multi-labels with 37 different classes so they were One-hot encoded using Multi-label binarizer (Figure 2) 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 2. Multi-Label Binarizer

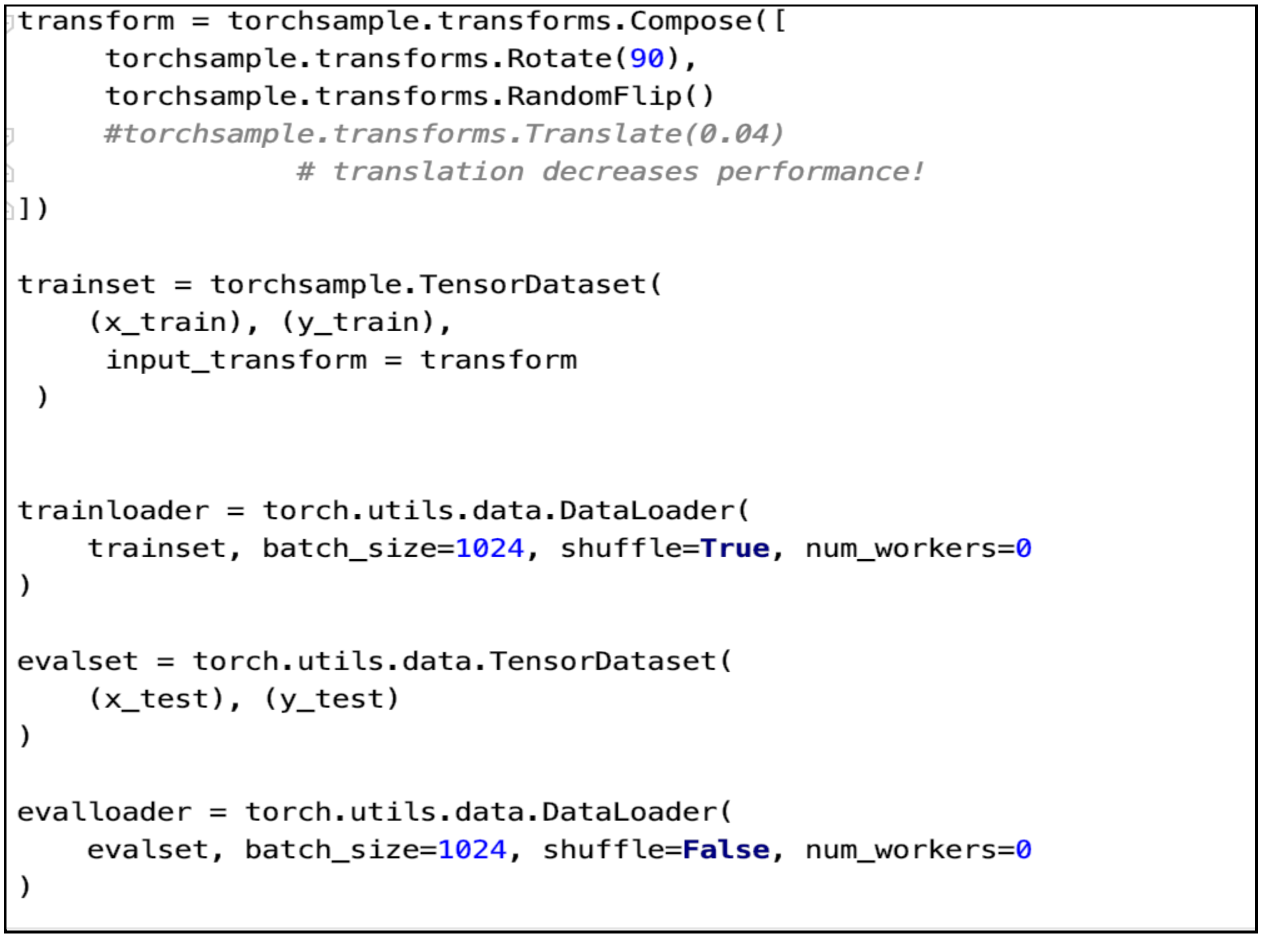
Further the data was split into training and validation set for modeling with respect to torch format to feed into CNN architecture.

Data Augmentation

Data augmentation technique was implemented to increases the range of recognition of images on the torch tensors just after data preprocessing using Torchsample package which is it's a data loader utility/module in PyTorch for image transformation. Then these images were feed into the trained CNN network. To transform the images, two types of augmentation functionality (Figure 3) was used

- Rotation: rotating the images randomly 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
)
```

Figure3. Data Augmentation

Conclusions

In Data preprocessing I tried to increase the size of the image like 120x120 or 28 x 28. of however the model was not improving. So, I decided to use 3 x 64 x 64 as image size.
After doing the data augmentation and on comparing the two CNN models (with and without augmentation) and the pretrained model which my fellow collaborators worked on it was found that the RMSE curve got smoother with data augmentation which determined it as the best fit model for our dataset.
Figure 4 below shows the RMSE loss on Train and validation loss

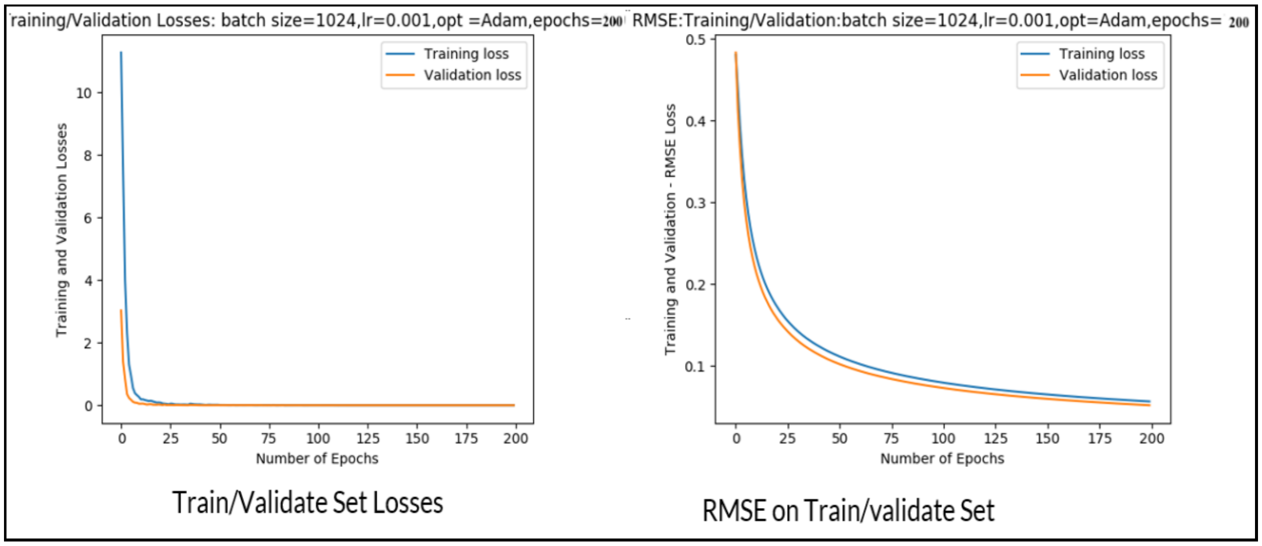


Figure 4. Training and testing losses and RMSE (Data Augmentation)

Summary

For future work I want to try more data augmentation techniques in the dataset with more computational power.

The percentage of the code that I found or copied from the internet was around 30-40 %.

References

- https://www.kaggle.com/helmehelmuto/keras-cnn
- https://github.com/benanne/kaggle-galaxies
- https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/