# Galaxy Image Classification

(Individual Report)

Author: Jyoti Sharma

DATS6203: Machine Learning II

The George Washington University

May 1, 2020

## Table of Contents

## Introduction

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.

## Shared Work

Mishkin, Tanvi and I researched together for finding the dataset, developing the problem statement, and finding the logic behind the work. We also read many articles to check how other people have worked in this kind of classification problem.

## Individual Work

In this section I tried building the CNN architecture which was utilized for training the network using PyTorch as framework.

### Network Architecture:

As we know, 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. In the project, after the data preprocessing, I divided the dataset in to training and validation sets, which further was converted in to torch format (Figure 1) for modeling.

```
# converting training and testing images into torch format
x_train = torch.from_numpy(x_train).float().permute(0, 3, 1, 2)
x_test = torch.from_numpy(x_test).float().permute(0, 3, 1, 2)
```

*Figure 1*. Torch Format and Conversion

I built basic 2 convolution layer network with batch normalization and maxpooling as shown in Figure 2.

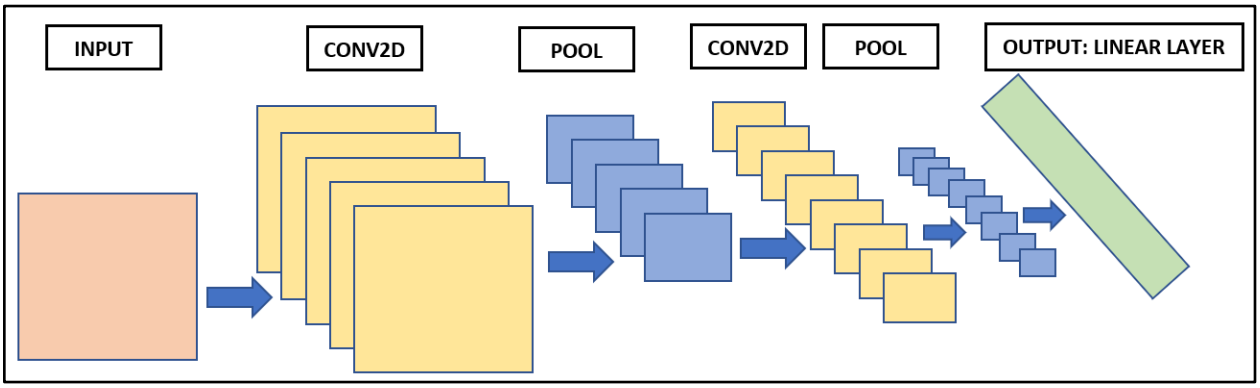


*Figure 2*. Network architecture

The algorithm implementation of the CNN network (Figure 3) that I developed utilized input shape of 64x64, feature maps 16 and kernel size 3 in the convolution layer 1. To reduce the sensitivities to initial weights I used batch normalization after first layer. I also applied maxpooling to extract and capture sharp, important and smooth features out of the image.

```
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 3*. CNN Network Flow

Similarly, convolution layer 2 was also built with feature maps 32, kernel size 3 and further followed by batchnorm and average pooling to output a fully connected linear layer with 37 classes. Further, model was defined with Adam as optimizer and MSELOSS as performance index. This model was then trained with batch size 1024 over 200 epochs on the training set and predicted on validation set for analyzing training and testing losses.

## Results

To classify the galaxy images for 37 different classes with their probabilities ranging from 0 to 1, I applied Sigmoid function at the output logit of our model. This way I evaluated the RMSE over each epoch for training and validation set. I also tried various batch sizes like 256, 512, 128, 32, 64 along with different learning rates like 0.01, 0.0001, 0.5. However, our model was not learning successfully. Only at a very specific values of batch size = 1024 and lr = 0.001 the model was learning and giving results.

Figure 4 shows the losses and RMSE over training and validation set. And it can be seen, there is significant lag between training and validation RMSE.

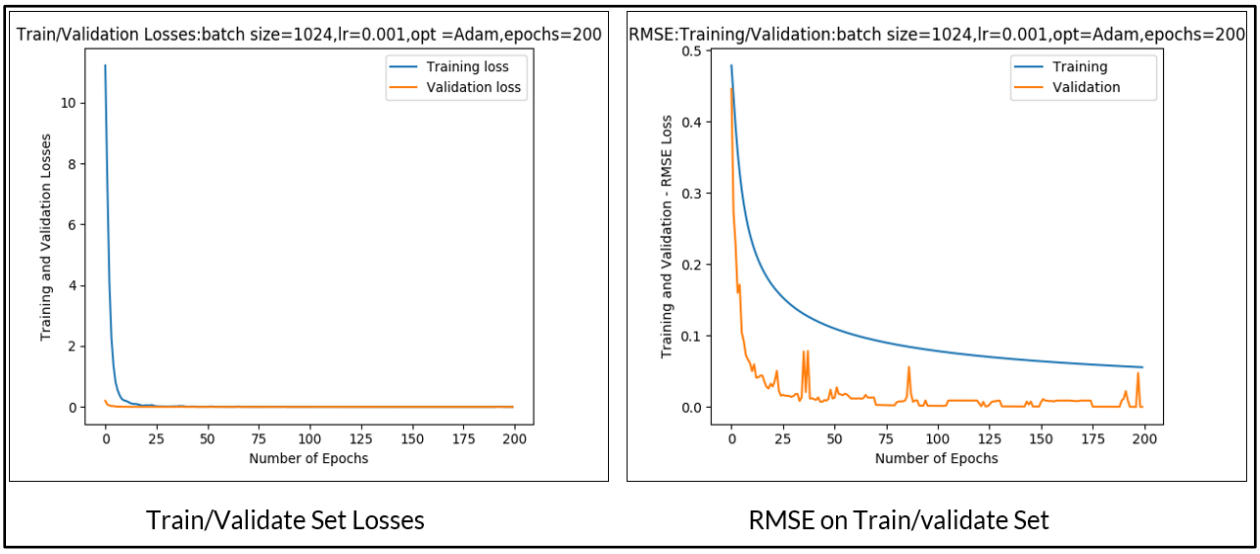


*Figure 4*. Training and testing losses and RMSE

In order to improve this difference, we further applied data augmentation using 'torchsample' package which gave us smooth curve with good RMSE results in our further findings.

## Conclusion

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 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.

Working with CNN is complex as we have to perform mathematical calculations between layers. These calculations helped me to understand what is happening under the hood. For future work, I would say more complex network architecture can be tried to check how does the model performs and output results.

## Code Reference

For individual work, 20% of the code I referred from the internet. I spent good amount of time in doing research and building CNN architecture for our problem statement.

## References


- https://github.com/amir-jafari/Deep-Learning
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://www.kaggle.com/helmehelmuto/keras-cnn

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