Project 1:

Image Classification using Feed Forward Neural Network

By Arunkumar Ramachandran

MTH 5320

10/9/2020

Contents

Table of Figures 3

Table of Tables 4

1. Introduction 5

2. Goal 5

3. Preprocessing Dataset 5

3.1. Normalize Function 5

3.1.1. Sklearn. preprocessing. normalize Function 5

3.1.2. Normalize Function using Min-Max 5

3.2. Resize the images: 6

4. Weight Initialization Methods 7

5. Network Architecture 7

6. Activation Function 7

7. Tune hyperparameters 8

8. Final Code 8

9. Discussions 9

Table of Figures

Figure 1 Normalization Using Min-Max Function 6

Figure 2 Test data (BGR) 7

Figure 3 Train data (BGR) 7

Figure 4 Tuning the hyperparameters 8

Figure 5 Final Accuracy obtained after training model 9

Table of Tables

Table1 Best values obtained on tuning the hyperparameters.. 8

1. Introduction

Image recognition refers to the task of inputting an image into a neural network and having its output label for that image. The label that the network outputs will correspond to a pre-defined class. There can be multiple classes that the image can be labeled as, or just one. If there is a single class, the term "recognition" is often applied, whereas a multi-class recognition task is often called "classification."

A subset of image classification is object detection, where specific instances of objects are identified as belonging to a particular class like animals, cars, or people.

2. Goal

The main goal of this project is to classify images according to their respective labels and train the model to obtain good accuracy.

3. Preprocessing Dataset

This section describes the assumptions and procedures used to set up the analysis and obtain solutions for them.

3.1. Normalize Function

In this project, two normalization methods were used. The two normalization methods used are:

- Sklearn. preprocessing. normalize function
- Min-Max Function

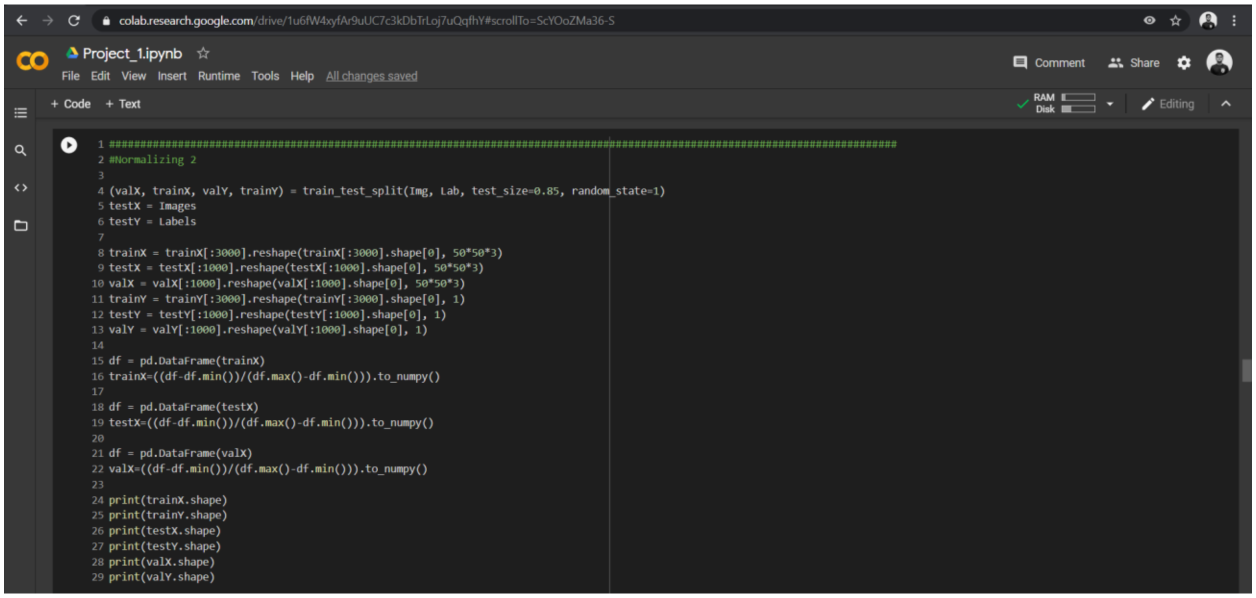
3.1.1. Sklearn. preprocessing. normalize Function

A normalized function scales the input vectors individually to the unit norm (vector length). The function can be depicted as:

```
sklearn.preprocessing.normalize(X, norm='l2', *, axis=1, copy=True, return_norm=False)
```

3.1.2. Normalize Function using Min-Max

The test and train set data were normalized by using the min-max function.

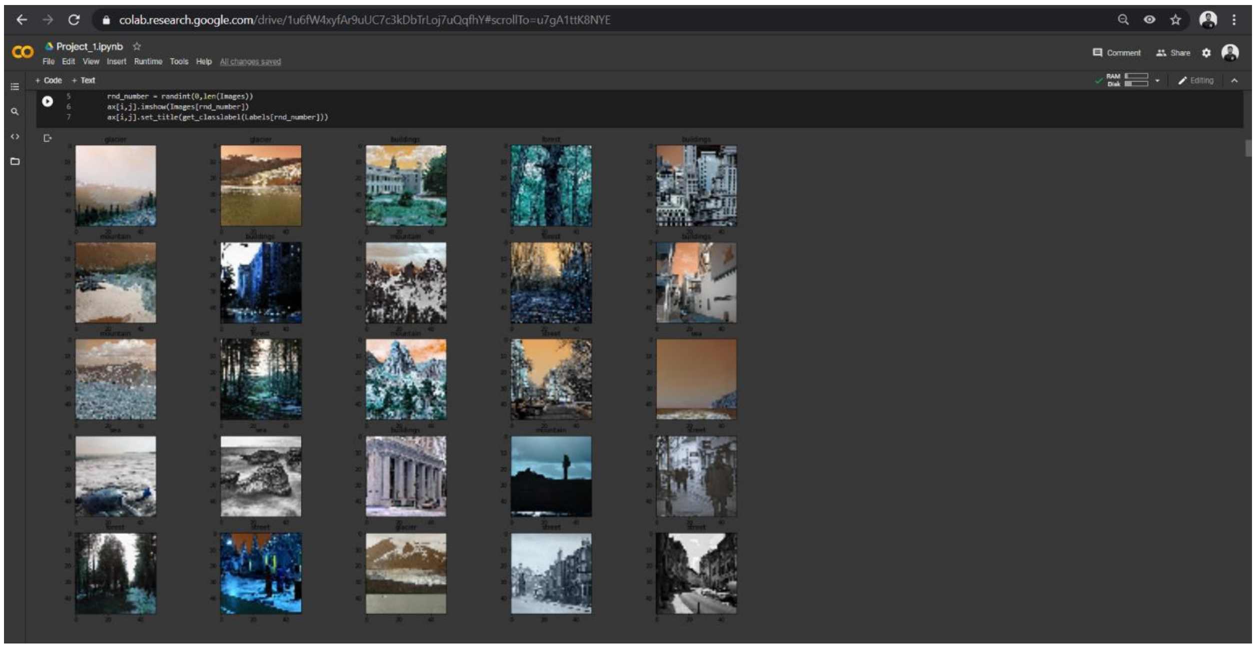


```
###############################################################################################################################
#Normalizing 2

(valX, trainX, valY, trainY) = train_test_split(Img, Lab, test_size=0.85, random_state=1)
testX = Images
testY = Labels

trainX = trainX[:3000].reshape(trainX[:3000].shape[0], 50*50*3)
testX = testX[:1000].reshape(testX[:1000].shape[0], 50*50*3)
valX = valX[:1000].reshape(valX[:1000].shape[0], 50*50*3)
trainY = trainY[:3000].reshape(trainY[:3000].shape[0], 1)
testY = testY[:1000].reshape(testY[:1000].shape[0], 1)
valY = valY[:1000].reshape(valY[:1000].shape[0], 1)

df = pd.DataFrame(trainX)
trainX=((df-df.min())/(df.max()-df.min())).to_numpy()

df = pd.DataFrame(testX)
testX=((df-df.min())/(df.max()-df.min())).to_numpy()

df = pd.DataFrame(valX)
valX=((df-df.min())/(df.max()-df.min())).to_numpy()

print(trainX.shape)
print(trainY.shape)
print(testX.shape)
print(testY.shape)
print(valX.shape)
print(valY.shape)
```

Figure 1. Normalization Using Min-Max Function

3.2. Resize the images:

The images contained in the train and test dataset have been resized from (152*152*3) to (50*50*3). The image for both the train and test dataset can be shown as follows:



```
        rnd_number = randint(0,len(Images))
        ax[i,j].imshow(Images[rnd_number])
        ax[i,j].set_title(get_classlabel(Labels[rnd_number]))
```

Figure 2. Test data (BGR)

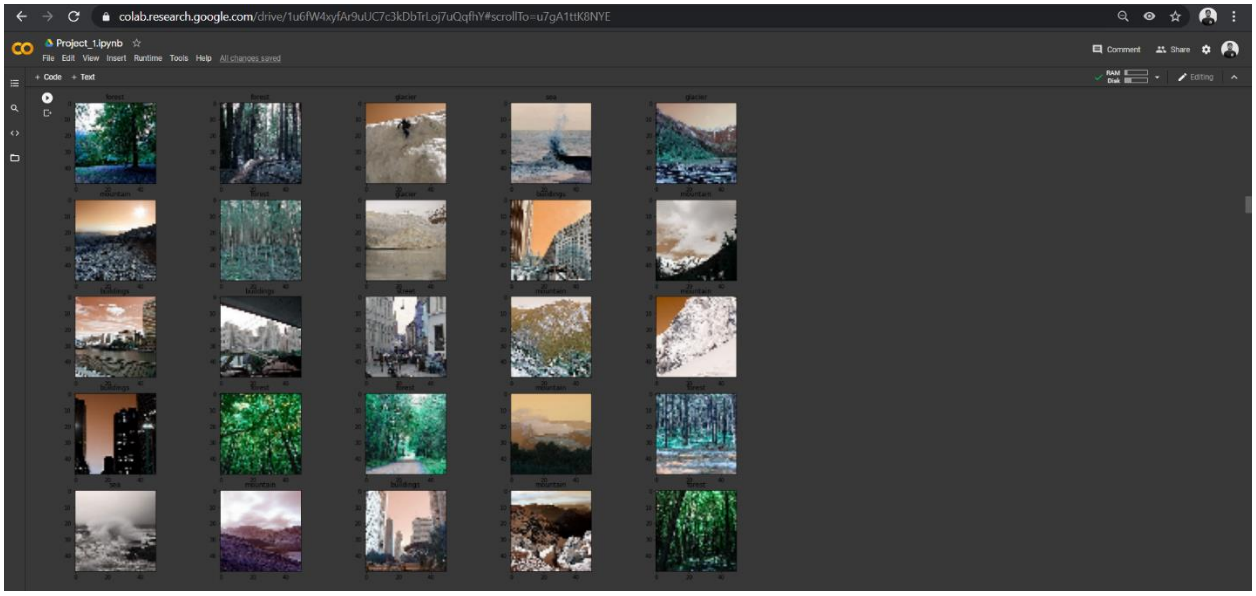


Figure 3. Train set data (BGR).

4. Weight Initialization Methods

Different weight initialization methods were used, and the one that gave the best accuracy was selected to tune the hyperparameters. The weight initialization methods used are shown in the following:

- Normal
- Uniform
- Lecum
- Glorot

Among these, Glorot showed better accuracy than the rest, and hence it was selected to tune the hyperparameters.

5. Network Architecture

Three Network architectures were used before tuning the hyperparameters. They are as follows:

- Network Archi1 = [(50*50*3), 62, 62, 6] ## One input, Two hidden layers, one output
- Network Archi2 = [(50*50*3), 64, 32, 18,6] ## One input, Three hidden layers, one output
- Network Archi3 = [(50*50*3), 64, 32, 42,52,6] ## One input, Four hidden layers, one output

Out of these layers, Network Archi 3 showed better accuracy than the other two. Hence, taking layer three into consideration, hyperparameters were tuned.

6. Activation Function

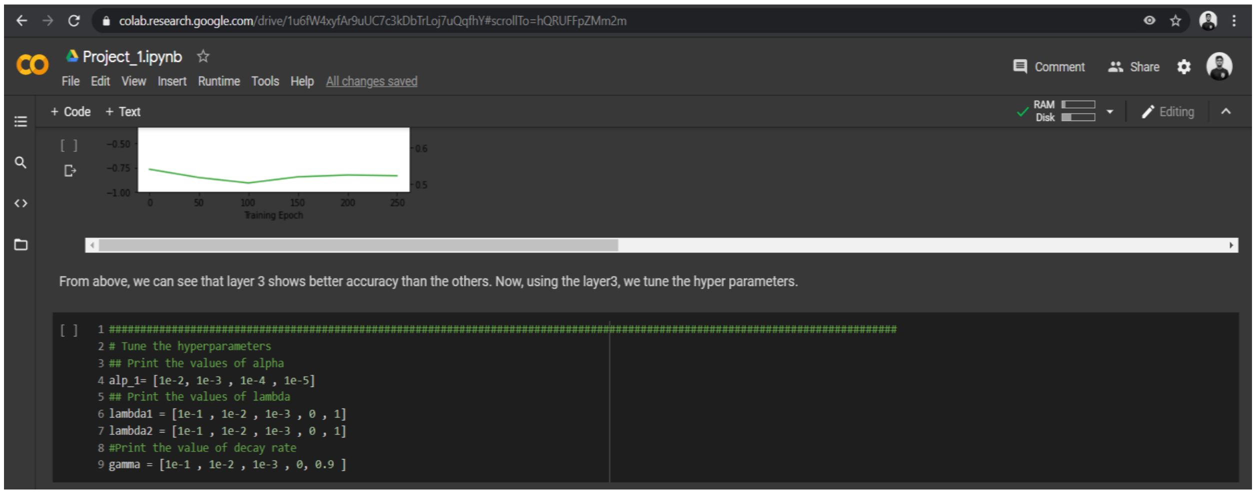
The activation function used in this project are as follows:

- ELU
- ReLU
- Sigmoid

Amongst these, ReLU, along with Glorat, showed better accuracy than the other two activation functions.

7. Tune hyperparameters

After selecting the right activation and weight initialization method, the hyperparameters were tuned. To tune the hyperparameters, specific values for lambda1, lambda2, alpha, and gamma were given to choose the best one out of these values depending on the accuracy they printed. The values used are as shown below:


colab.research.google.com/drive/1u6fW4xyfAr9uUC7c3kDbTrLoj7uQqfhY#scrollTo=hQRUFFpZMm2m
Project_1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
Training Epoch

From above, we can see that layer 3 shows better accuracy than the others. Now, using the layer3, we tune the hyper parameters.

```
##############################################################################################################################
# Tune the hyperparameters
## Print the values of alpha
alp_1= [1e-2, 1e-3 , 1e-4 , 1e-5]
## Print the values of lambda
lambda1 = [1e-1 , 1e-2 , 1e-3 , 0 , 1]
lambda2 = [1e-1 , 1e-2 , 1e-3 , 0 , 1]
#Print the value of decay rate
gamma = [1e-1 , 1e-2 , 1e-3 , 0, 0.9 ]
```

Figure 4. Tuning Hyperparameter

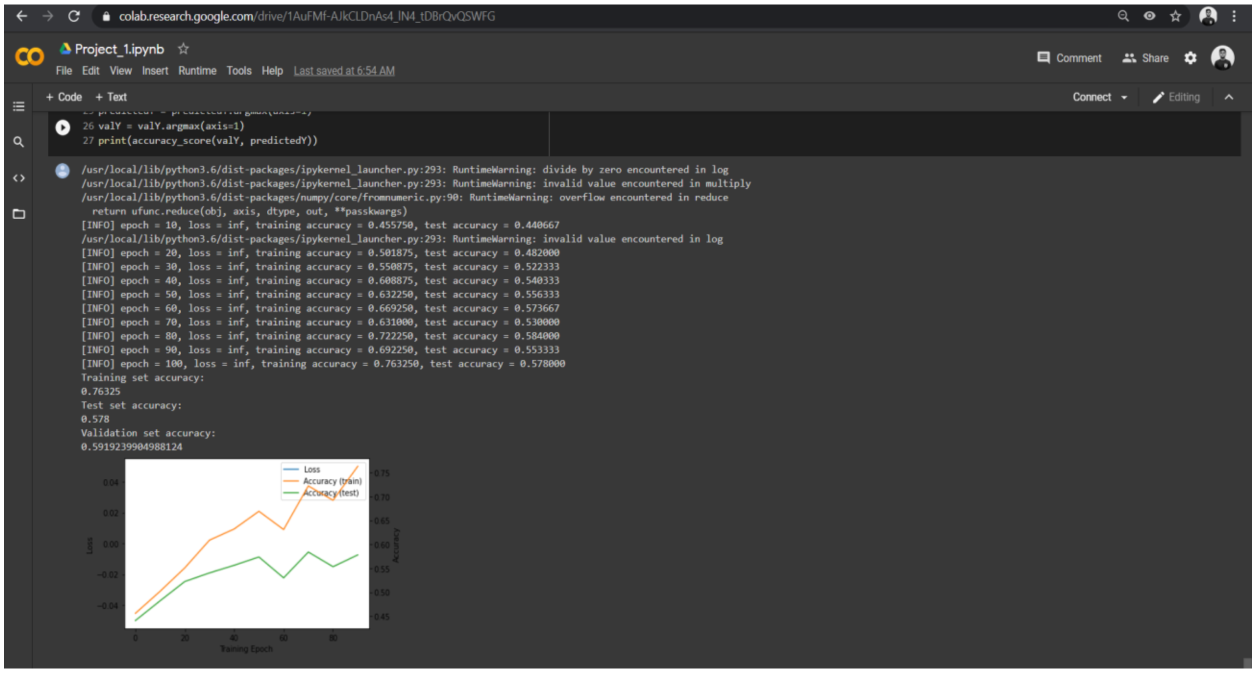
The best values selected can be shown in the following table

Lambda 1	0.1
Lambda 2	0.01
Alpha	0.001
Gamma	0

Table 1 Best values obtained on tuning the hyperparameters.

8. Final Code

Using the above values, and with the activation and weight initialization methods selected, a final code is drafted to check the accuracy of the test, train, and validation. To run the final code, more train set data was used. The accuracy obtained can be shown in the following figure:



```
valY = valY.argmax(axis=1)
print(accuracy_score(valY, predictedY))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:293: RuntimeWarning: divide by zero encountered in log
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:293: RuntimeWarning: invalid value encountered in multiply
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:90: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
[INFO] epoch = 10, loss = inf, training accuracy = 0.455750, test accuracy = 0.440667
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:293: RuntimeWarning: invalid value encountered in log
[INFO] epoch = 20, loss = inf, training accuracy = 0.501875, test accuracy = 0.482000
[INFO] epoch = 30, loss = inf, training accuracy = 0.550875, test accuracy = 0.522333
[INFO] epoch = 40, loss = inf, training accuracy = 0.608875, test accuracy = 0.540333
[INFO] epoch = 50, loss = inf, training accuracy = 0.632250, test accuracy = 0.556333
[INFO] epoch = 60, loss = inf, training accuracy = 0.669250, test accuracy = 0.573667
[INFO] epoch = 70, loss = inf, training accuracy = 0.631000, test accuracy = 0.530000
[INFO] epoch = 80, loss = inf, training accuracy = 0.722250, test accuracy = 0.584000
[INFO] epoch = 90, loss = inf, training accuracy = 0.692250, test accuracy = 0.553333
[INFO] epoch = 100, loss = inf, training accuracy = 0.763250, test accuracy = 0.578000
Training set accuracy:
0.76325
Test set accuracy:
0.578
Validation set accuracy:
0.5919239904988124
```

Figure 5. Final Accuracy obtained after training model

As seen in the above figure, the accuracy obtained for the test, train, and validation are:

- Train set – 0.76
- Test set – 0.578
- Validation Set – 0.591

9. Discussions

As seen above, the accuracy values touched 58 percent. This maybe because of using feed-forward neural network architecture. By using a better deep neural network such as CNN, there may be a chance to improve our accuracy values than the ones obtained above.