SCAAI 2nd Round Project Analysis

Anindyadeep Saanigrahi

Table of contents:

1. **Project overview and problem statement.**
2. **Analysis of handling the data part.**
3. **Model making and results of the models.**
4. **Instruction for running the project for testing.**
5. **Conclusion.**
6. **References.**

Project Overview and problem statement

The problem statement I have chosen for this project was to create an image classifier that would classify the images of the natural disasters. (kaggle link). This maximum part of this project is being done on Google Colab (due to its GPU) and some data handling parts are done on Jupyter Notebooks.

Analysis of hanelling the data part.

In this part, I will be explaining the approaches, which I have used here to handle the data in order to get some good and generalized results from the model. There are a total 2 approaches in which the first approach didn't give any satisfactory results. The second approach gave me more generalized and satisfactory results w.r.t the metrics used to test the model with unknown test dataset customly made and labelled by me, with 25 data points.

Approach 1:

Assumption: Here I have assumed that I have in total 12 classes to classify, as I have considered the subfolder as one single class. And so here are the subfolders that I have considered initially with their number of datapoints.

Class folder	Initial number of data points
Drought	201
Earthquake	36
Human Damage	241
Infrastructure	1418
Landslides	456
Urban fire	419

Wildfire	514
Non Damage Human	120
Non Damage Building	4572
Non Damage Wildlife forest	2271
Sea	2274
Water Disaster	1035

As we can see from the above, the distribution of the data is heavily unstable and unbalanced. And training a model with this type of data will get hugely biased over one class, with the highest number of the data points viz: Non damage Building.

Approach

So here I first went to go for image augmentations. I first created a full python classes, that would do the following tasks:

1. Compute the number of datapoints with the most number of datapoints. (here 4572)
2. Now I will take a minority class folder and calculate by how many times the data points should increase to be similar to the one with the max number of data points. E.g. Class **wildlife** has 514 data points which is approximately 8 times less than the one with max data points.

3. So after calculating that scaling factor, it will go under deep augmentations (which is simple pytorch augmentations with several options sampled under a probability to get augmented). So from the above example, each of the images will get randomly augmented 8 times the actual image and will be saved in the similar folder. And this is how we are up-sampling the data points for the minor folders using this technique.

4. If there is any reason that the minority class folder gets too many images, then some images from that folder are randomly deleted such that the distribution of the data points is normalised everywhere.

Similarly this is done for all other minority folders, and these are the results we are getting from them after auto-augmentations and image generation.

Class folder	Number of data points
Drought	4672
Earthquake	4464
Human Damage	4560

Infrastructure	5672
Landslides	5016
Urban fire	4386
Wildfire	4626
Non Damage Human	4560
Non Damage Building	4500
Non Damage Wildlife forest	5013
Sea	4958
Water Disaster	5175

Results after model building in approach 1:

In one word, it's below average or poor. I have tried several transfer learning models. But all of them were giving me the same kind of results. And the model is biased for a single class Non Damage Building. The model was quite overfitting itself. Which was clear from the train and validation metrics.

Possible reason of the fail of approach 1:

As the distribution of the data was too low for some of the class folders, augmenting them might increase the number of significant features in order to make a good classifier. But for some folders, the number of randomly augmented images were higher than the actual number of images, so during the validation and testing of the actual images, there might be a spatial difference between the features of those two, which we can say as good and bad features. So these might be some possible reasons for the model getting overfitted and becoming a garbage model.

Some colab links for the above approach

APPROACH 1 COLAB1 APPROACH 1 COLAB 2

Approach 2

Assumption: As from the above approach, it was clear that with this amount of the actual number of datapoints it is near to impossible to classify 12 classes of the images. So this time, I am assuming that the task is to classify 6 classes of the images instead of 12 classes, where I

merged the subfolders as one single class. So in this case the distribution of the folders are as follows:

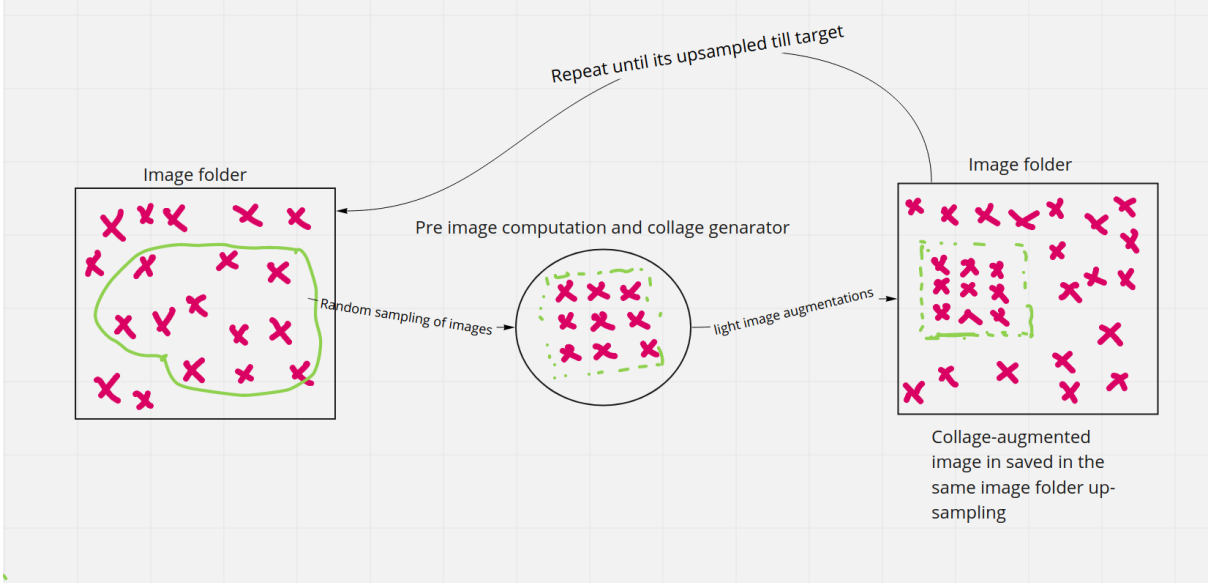
Class Folder	Initial number of datapoints
Damages Infrastructure	1454
Fire Disaster	933
Human Damage	241
Land Disaster	657
Non Damage	9237
Water Disaster	1035

Approach

So implementing this approach was very much interesting and fun. This approach follows this sort of steps as algorithm

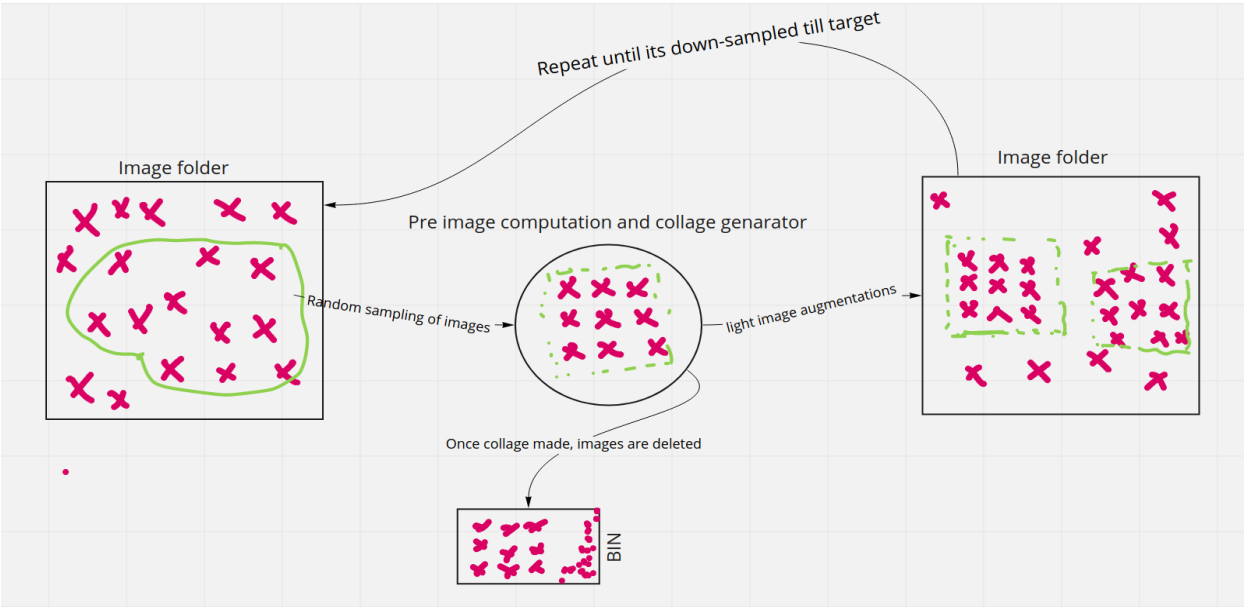
1. First select the a number of samples that would fit for all the folders (here i chosen a sample ranging from 1000-1200 data points)
2. After this we are going to upsample those folders which are deficient of data points and downsample the data points of those folders that contain an excess number of datapoints.
3. **Process of upsampling**:
    a. Once the sample range is being selected, then we look at the initial number of the data points in the current folder. Let's say the number of the data points is 200.
    b. Upsampling technique process is little different here. Here instead of making random augmentations of the images, we are taking any random 9 images from the folder and making a (3 x 3) image collage of that. And then passing that collage into very light augmentations (which just consist of flipping vertical, horizontal, resizing, converting to tensor) (As deep augmentation could change the spatial geometry of every single image of collage, resulting in getting less significant features.)
    c. Also a point to note that before going for collage, we first resized each image to (1024 x 1024) such that the final collage image is (3072 x 3072) size. And in augmentation we are resizing it to (224 x 224) images. The only reason for this is that during making of the collage of the images, the clarity of each image as a feature could decrease or lower if we take original size images. So while resizing it to 1024 then converting to 224, the spatial clarity remains somewhat the same.

Illustrations of up-sampling of the data



4. **Process of the downsampling**
    a. Suppose we have 9000 data points in the folder and we want to downsample it to 1000 images. So in that case, we randomly take 9 images and make a collage out of it, maintaining the same conventions of making the collage , explained in point (3-c). But here is a catch, once the collage is done, we are deleting the images forming the collages from the folder. So finally the number of the images gets downsampled from 9000 to 1000 with no redundant images.

Illustrations of down-sampling of the data



So after this process the distribution of the data is shown below in the table

Class Folder	Number of datapoints
Damages Infrastructure	1205
Fire Disaster	933
Human Damage	940
Land Disaster	1157
Non Damage	1228
Water Disaster	1035

Results after model building in approach 2:

This time the results were much more generalized and satisfactory in terms of the metrics based on classification like precision, recall, f1 score, accuracy. Also there were satisfactory results in the confusion matrix too.

Colab link for the above approach

COLAB FILE FOR THIS APPROACH

The data generated is being saved as a zip file for the further training of the models based on this data.

Model making and results of the models.

During the whole process I have taken several models into consideration like DenseNet201, ResNet50, ResNet152, VGG19, VGG16. VGG19_bn, GoogleNet, etc. But here I will show the process and results of the top 3 performers (based on training metrics).
So here is the detailed analysis of the models metrics during the time of the training

Background knowledge
Python version: 3.8
Library used: PyTorch
Dataloaders of 128 batches each
Epochs 5

Learning rate: 0.00175
Optimizer: Adam
Scheduler: StepLR

Table 1: Training metrics of the model

Model	Loss	Accuracy	Precision	Recall	F1-score
ResNet152	0.4437	0.8651	0.8658	0.8658	0.8658
VGG19	0.3506	0.8799	0.8801	0.8801	0.8801
VGG19_bn	0.4568	0.8439	0.8443	0.8443	0.8443

Table 2: Validation metrics of the model

Model	Loss	Accuracy	Precision	Recall	F1-score
ResNet152	0.4783	0.8552	0.8541	0.8541	0.8541
VGG19	0.3505	0.8798	0.8801	0.8801	0.8801
VGG19_bn	0.4244	0.8638	0.8638	0.8638	0.8638

Table 3: Test metrics of the model

Model	Accuracy	Precision	Recall	F1-score
ResNet152	0.44	0.44	0.44	0.44
VGG19	0.76	0.75	0.75	0.75
VGG19_bn	0.08	0.08	0.08	0.08

From the above 3 tables it is very much clear that VGG19_bn, and ResNet152 are overfitted, whereas VGG19 has been able to achieve the just right state.
NOTE: All of the models were working pretty good on the dataset when they were running on Google colab (GPU) on the same dataset, but when they were run on the CPU offline then

RESNET152, VGG19_bn had somehow lost their properties. Whereas VGG19 existed as the one working both well in GPU and CPU. I don't know how this has happened.

Google Colab files for the three models training:

VGG19 model , VGG19_bn model , ResNet152 model

Instruction for running the project for testing.

Project file structure:

```
-- <SCAAI_Drive_Anindyadeep_Sannigrahi_PS3>
       |_______ code
                   |___ progress_test.py
                   |___ results.csv
                   |___ run.py
       |_______ Comprehensive Disaster Dataset(CDD)
                   |__ Damaged_Infrastructure
                   |__ Fire_Disaster
                   |__ Human_Damage
                   |__ Land_Disaster
                   |__ Non_Damage
                   |__ Water_Disaster
       |_______ IPYNB
                   |__ FINAL_APPROACH_FILE_MAKING.ipynb
                   |__ RESNET152_final224by224.ipynb
                   |__ VGG19_bn.ipynb
                   |__ VGG19.ipynb
       |_______ models
                   |__ RESNET152.pth
                   |__ VGG19_bn.pth
                   |__ VGG.pth
```

```
        |_______ rough tests
                    |__ model testing-resenet152.ipynb
                    |__ model testing-vgg19_bn.ipynb
                    |__ model testing-vgg19.ipynb
        |_______ test_folder
        |_______ final.zip
        |_______ Readme.md
        |_______ Requirements.txt
```

Here **Last Trial** is the actual master folder under which all the works will be done.

STEPS FOR CONFIGURING THE TEST FOLDER

There are two ways in which a user can run the model to test the images.

1. You can either go to the **test_folder** and copy paste the images that are being collected there and our **run.py** file will process those images automatically and will provide the outputs.

2. You can also remove the pre-existing **test_folder** and paste the required folder of the user that will be used for the testing purposes, just **rename** that folder to **test_folder**. But the folder must be placed in the main folder**<SCAAI_DRIVE_Anindyadeep_Sannigrahi_PS3>** removing the old **test_folder** folder. Otherwise it will not work.

Running the code commands

I have made my own test images which are in the **test_folder** that I have used to analyse my model. You can just see the accuracy of the model on the basis of that test data provided by me by going to the **code** subfolder and there you can run this following code as shown below:

```
python3 -c 'import os; import run; print(run.get_accuracy_test_results())'
```

If any how this above command doesn't work, then go to the **code** subfolder and run:

```
python3 get_accuracy_test_results.py
```

For running the model for an unknown dataset provided by the user, can be run through this command below. **NOTE**: The folder **test_folder** must be at the right place.

```
python3 -c 'import os; import run; path =
str(os.getcwd())[:-4]+"test_folder"; print(run.get_test_results(path))'
```

If any how this above command doesn't work, then go to the code subfolder and run:
```
python3 get_test_results.py
```

NOTE: All the work can be easily done, if folder structures are not changes and at the same time, test datasets are kept in the **test_folder** folder.

Conclusion

The total life cycle of this project was really interesting. The future work could be done in this case to use GANs in order to generate images for the datasets but that approach is very much computationally less feasible. Though generating more images with the above techniques with good augmentations may increase the model's accuracy more. I hope the model could pass other test datasets too.

References

Making collages using python

PyTorch tutorial websites