POTHOLE DETECTION USING DEEP LEARNING

Prepared for

CSCI6505 – Machine Learning

Initial Implementation

Prepared by

Janvi Patel, B00863421
Jeyanth Kishore Ramasamy, B00875285
Robinder Dhillon, B00876516
Vishal Sancheti, B00877378

February 27, 2021

1. Introduction

The number of vehicles on roads has increased a lot in this modern era and so as the count of accidents. Some of the accidents in roads happens due to the existence of potholes in the road. This project deals with identifying the potholes in the road and thereby alerting the drivers to reduce the speed in advance. It is even possible in future to invent a new technology which automatically reduces the speed based on the existence of potholes on the road. The algorithm will be trained with large amount of data images of roads with potholes and normal roads without potholes. When the algorithm achieves high accuracy then they can be used in vehicle in which the data from sensor which captures the images of the road will be fed. Then those data are processed to identify the potholes on the road. This will be very helpful in reducing the road accidents.

2. Dataset

Dataset needed for this project are collection of images which can be used to detect potholes. We found few interesting datasets as below:

- https://www.kaggle.com/virenbr11/pothole-and-plain-rode-images
- https://www.kaggle.com/atulyakumar98/pothole-detection-dataset
- https://www.kaggle.com/sovitrath/road-pothole-images-for-pothole-detection

These datasets will be processed to generate uniform dataset with required features for the project. The above datasets will be split into two segments as training data and test data and later they will be fed to algorithm to fine tune it.

3. Implementation Plan

The initial implementation will focus on Understanding the data and processing it to generate a data set useful for the project.

The steps involved in this are:

1. Image selection: Depending on the number of grayscale images already in the dataset, we will

a) Delete the grayscale images - if the percentage of such images is low enough (3%)

b) Convert RGB images to grayscale - if the percentage of grayscale images is significant.

2. Image labeling: We will first take the image set and create a dataset where we will have the image name and if that image has a pothole or not. If the image contains a pothole, the corresponding value will be 1. If it does not have a pothole, the value will be 0.

3. Image resize (dimensions) fixing: In this step we will set the dimensions of each image. Since we will be using a simple Multilayer Perceptron, the number of inputs will be fixed. Therefore, it is required that all images have the same dimensions (m by n for example), we will modify the images (crop, stretch, resize etc) so that all of them have the same dimensionality.

4. Image to numerical data conversion: In this step we will convert the image to numerical data. Depending on the 1st step

a) Since all the images are colored, they will have three channels (R, G, B) with each pixel in a channel having a value in range [0,255]. We will read the pixels row-by-row and channel-by-channel, giving us a list with m*n*3 values. These 3*m*n values will then be the input to our multilayer perceptron.

b) Since all the images are grayscale, they will have only one channel with each pixel a value in range [0,255]. We will read the pixels row-by-row and channel-by-channel, giving us a list with m*n values. These m*n values will then be the input to our multilayer perceptron.

Using the above data, we will train our model to detect whether the road has a pothole or not.

Final Implementation:

The final implementation will be a Trained model using MLP and CNN neural network which will be capable of detecting potholes and if possible, tag potholes on image using boundary or heatmap.

3. Anticipated Challenges

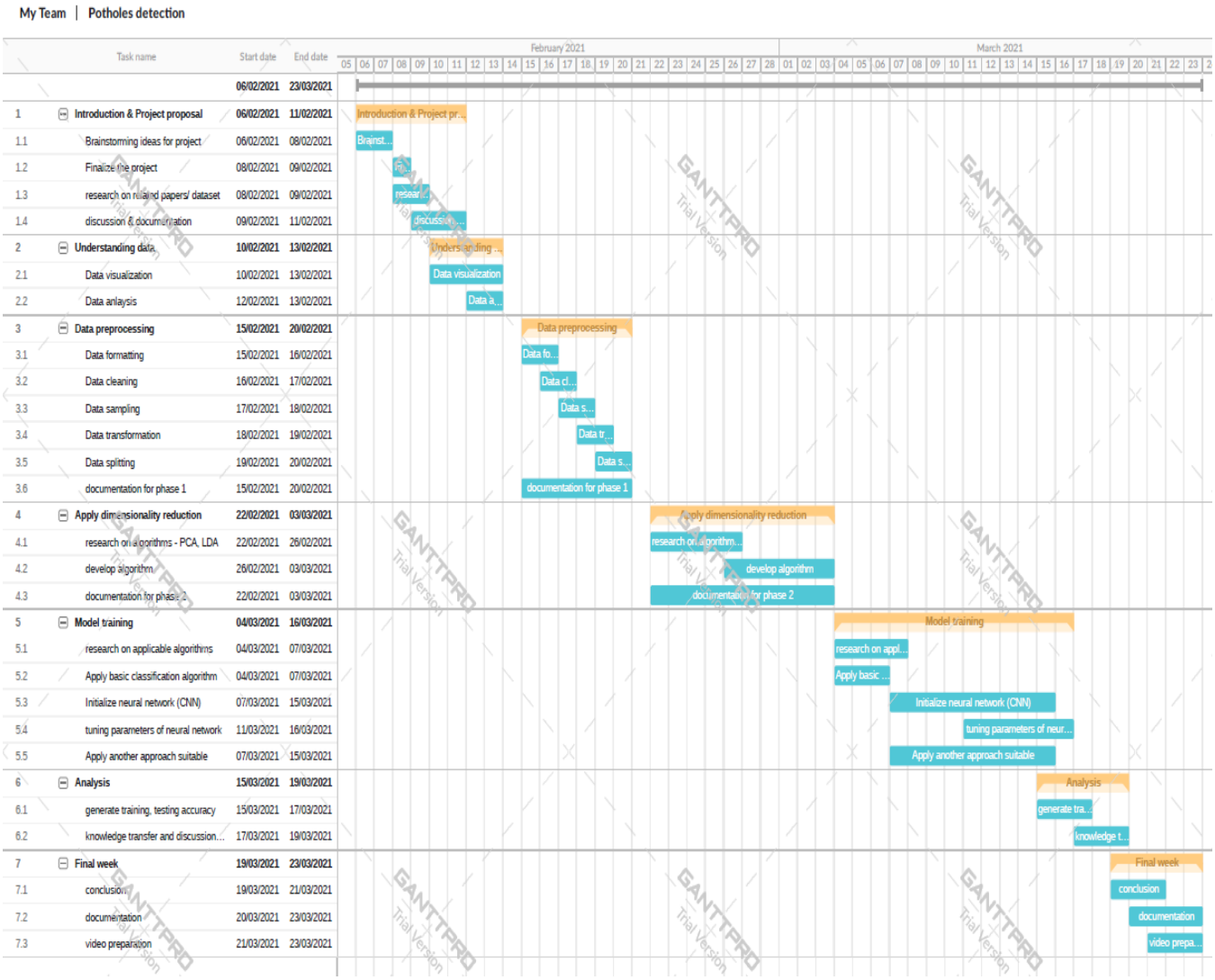
The challenges involved in this project are Data Pre-processing, Training Model, and Visualization and Analysis of results.

Currently, the collection and pre-processing of data are an easy part of the project as a significant source of the data is from Kaggle, and pre-processing needs only a few transformations. The major challenge for us is choosing neural network for our training model. We are currently only familiar with Multi-Layer Perceptron and are aware there are advanced neural networks available such as CNN, AlexNet, VGG, etc. As a backup plan, we wish to train our model using CNN neural network if MLP results lack accuracy.

Once the model is trained, it is straightforward to detect if an image has a pothole or not. Still, visualizing the results and detecting the location of the pothole on image and tagging it is which we are less confident about. We also don't know if we could tag the pothole's location using boundary or heatmap as this depends on the results from previous tasks.

4. Timeline and Milestones

The details are attached as an A**nnexure I: Gannt Chart** along with this report



5. Roles and Responsibilities

Every person will be responsible for the completion of the project, but responsibilities of major tasks are divided are as follow:

Responsibility	Janvi.P	Jeyanth.R	Robinder.D	Vishal.S
Data gathering and Formatting				Yes
Data cleaning and sampling	Yes			
Data transformation		Yes		
Data Splitting			Yes	
PCA	Yes			Yes
LDA		Yes	Yes	
MLP: Model Training		Yes	Yes	
CNN: Model Training	Yes			Yes
MLP: Analysis and Visualization		Yes	Yes	
CNN: Analysis and Visualization	Yes			Yes

6. Immediate Tasks

Currently during the time waiting for approval of the project, we planned to create a boilerplate code for image detection. We will do various research for image identification and converting them into data that could be fed to the algorithm. This would be the primary step required for the project. We will further do analysis on various papers published in this topic to get an overview of what is to be done and what new things could be implemented apart from the existing technologies.

7. Initial Implementation

As suggested by professor Sageev Oore, we decided to train our model using various CNN methods, neglecting the process of training model with MLP techniques. Initially, we downloaded a dataset from Kaggle(https://www.kaggle.com/amanrajput27/pothole-detection-by-cnn-pytorch). The first task was data preprocessing so that it is ready to run our training model on. The dataset was checked manually to see if the percentage of Grayscale images was greater or less than 3% (threshold selected). If the grayscale images were more than 3%, all the images were to be converted to Grayscale. Since there was only one image in the dataset, which was Grayscale, it was deleted.

Then those images were placed in a common drive, so that all the members of the project can have access to it.

Image Pre-processing:

Since all the images in dataset were in jpg format, we wrote the code to access those images whose format is jpg. This condition was used so that if some other format file (.exe etc.) is pasted in our dataset folder, it can simply be ignored, and the processing can continue. Care will be taken to include all the extensions of the image files so that a wide format of images can be processed. Afterwards, we wrote the code for converting the images to RGB. During initial implementation, it was seen that all the images were not in RGB mode. So, we used the **PIL** package and we imported **Image** to achieve our goal.

Image Resizing:

After converting all images to RGB we wrote the code for image dimension fixing. One of the reasons for image fixing is so that model can compute faster. As of now, resizing would have sufficed, so we wrote the code to resize the image. We ran a for loop, traversing each image and resizing it. The above steps were implemented on the folder containing pothole images as well.

We also explored the ways to compress the images by reducing the pixel color from 256 to 16. This will reduce the size of the image and the pixel intensity. As of now we have decided to go with original pixel intensity of the images, and we will be compressing the image based on pixel if our algorithm takes more time to train.

Creating Training and Test Datasets:

Then the images in dataset are split into training and test datasets. We decided to train our algorithm with the training data set and then we will be testing our algorithm with the test datasets. This will help to increase the efficiency of the algorithm and we can make sure that the algorithm gives best result. Almost 70% of the given dataset are considered as training datasets and the rest are test datasets. We will be choosing random images from the given data set and moving them to new folder depending upon our requirements. By this way, we have split the given dataset into two separate training and test datasets. From the training datasets, validate datasets has been created. By using this validate data we can test and fine tune our algorithm. It helps us to minimize the loss function of our algorithm

Conversion to Numeric Data:

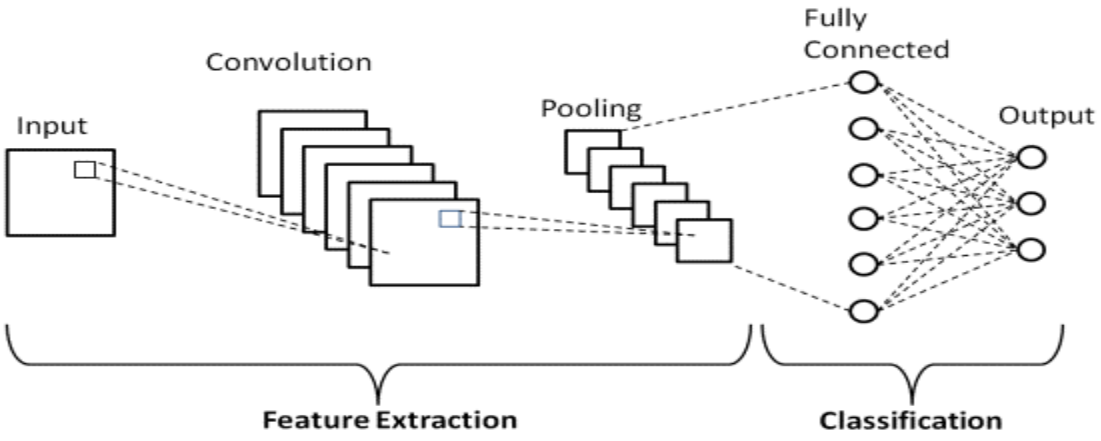
The processed images are then converted into numeric data. Since all the images in the dataset are colored, they have three channels (R, G, B) with each pixel in the channel having value in the range 0-255. We cannot use the image data directly to the model, so they are converted into pixel numeric data. Since our images are in the dimension 200*200 (m*n), the output of the numeric conversion of image gives us an array of dimension 200*200*3 (n*m*3), where 3 is the number of color channels.

Since colored images takes more time for training, we have converted them to Grayscale image during the initial implementation of the project. We have also converted dimension to 50 * 50 for easy training and implementation of the model. Later in our project we will be using colored images with high dimension. We have used torch.floatTensor of python for conversion of image into numeric data. Then to normalize the data, the numerical values are divided by 255 to get the values within range of 0-1. After converting into numeric data, the result labels (y values) are appended to the numeric dataset. After this, we created a dictionary to store all the values in it. The dictionary was given the key as a simple index starting from 1**.** However, the value of the key is a list which contains the image object as well as a label (0 or 1). Label 0 means that the image does not have a pothole and label 1 means that the image has a pothole. A for loop was used to traverse the resized image numeric dataset and corresponding entries were added in one dictionary.

Then all the numeric data of potholes and normal images are merged into single training dataset and those numeric data of images are shuffled. Similar process is carried out for the test dataset and the result is also shuffled.

8. Kaggle Base Code

Initially we have decided to take the code base that is already available in the Kaggle (https://www.kaggle.com/amanrajput27/pothole-detection-by-cnn-pytorch). They implemented basic CNN to detect the pothole images. The architecture of the basic CNN model is displayed below.



We used their implementation to understand the working of the CNN. After this initial implementation, we have decided to use various model available in CNN and develop algorithm for each model. Based on the efficiency of each model, we will decide the suitable model for our project. The individual contribution of models in CNN is given below.

Vishal Sancheti – AlexNet
Janvi Patel – VGG
Robinder Dhillon – RestNet
Jeyanth Kishore Ramasamy – LeNet

For learning purpose, we have used the CNN model code base mentioned above. Also, we used their method to feed the algorithm with the training and test data. It is seen that training data gives us a loss function of about 0.348. Test data gives us the accuracy of about 40%. We will be tuning the algorithm in the next stage of the project and improve the efficiency of the current model. Then we will try to build a new model in AlexNet, LeNet, RestNet, and VGG. We will fine tune our respective model to achieve high accuracy and at later stage of our implementation we will decide upon a model that suits our project.

Kaggle Code Explanation

The code first defines a network class which creates 3 convolution layers using the nn.Conv2d method. This method applies 2D convolution over the input signal. [1]

self.conv1 = nn.Conv2d(1,32,5)

This means that input is 1 image, 32 output channels and 5X5 kernel.

The next conv. Layer will have input as 32 because the output of previous layer is 32.

After this, we need to flatten the image, which means converting the data in 1-D array for input to next layer. Once the image is flattened, softmax formula is applied combined with logarithm(log_softmax). These two functions (softmax and logarithm) if done separately, can be numerically unstable, so we use this method.[2]

For training the model, we use Adam algorithm which is an optimization algorithm provided by the torch.optim class.[3] For training the data, the code uses cross entropy loss. However, we can even use squared error loss function. We then separate the features and labels. We also select the percentage of data that we need to use as training and test data. A batch size is then selected and negative log-likelihood loss function is called on the batch. After calling the function, the gradients are computed and then optimizer.step() iterates through tensor and upgrades the values that are supposed to be updated.

8. Bibliography

[1] Kaggle, Web portal, “Datasets” 2021, Accessed on: 10 February 2021 [Online]. Available: https://www.kaggle.com/datasets

[2] S.Abirami, P.Chitra, ScienceDirect Website, “Multilayer Perceptron” 2020, Accessed on: 10 February 2021 [Online]. Available: https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron

[3] K.Kask, University of California, Irvine, Document, Machine Learning and Data Mining, “Multi-layer Perceptrons & Neural Networks: Basics” 2021. Accessed on 11 February 2021 [Online]. Available: https://www.ics.uci.edu/~kkask/Spring-2018%20CS273P/slides/08-mlpercept.pdf

[4] Prabhu, Medium, Blog, “Understanding of Convolutional Neural Network (CNN) — Deep Learning” March 4, 2018. Accessed on 11 February 2021 [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[5] T. Folkman, TowardsDataScience, Blog, “How To Tag Any Image Using Deep Learning” May 16, 2020. Accessed on 11 February 2021 [Online]. Available https://towardsdatascience.com/how-to-tag-any-image-using-deep-learning-84a0dc2e03c2