CS6301: Final Project Report

https://github.com/Milind21/AdvNLP

Choudhary, Milind
MXC210096

Honakeri, Deepak Kumar P
DPH200003

Singh, Arpit
AXS210204

1 Abstract

The aim of this project is to develop a text classification model for plant diseases using a newly created dataset obtained from scraping relevant information from Wikipedia pages. Specifically, the goal is to accurately predict the type of plant disease based on textual descriptions, which can contribute to the development of effective solutions for identifying and managing plant diseases. Additionally, the project aims to compare the performance and accuracy of the text-based classifier with an image classifier, in order to determine the strengths and limitations of each approach for plant disease classification.

2 Introduction

Plant diseases have been responsible for causing famines throughout history, leading to devastating consequences for entire communities and nations. However, with the advancement of technology, it has become easier to detect and diagnose plant diseases in their early stages, minimizing their impact on vulnerable communities and ecosystems. The loss of crop yield due to plant and crop diseases [2] [5] results in significant wastage. This issue is particularly relevant in countries where the economy is heavily dependent on agriculture [1].

Text classification can aid in early detection. Natural language processing techniques, such as the automated text classification algorithms, can analyze large volumes of text data to recognize patterns and correlations related to plant diseases. These algorithms can scan various text sources, including scientific publications, news articles, and social media posts, to identify mentions of plant diseases, their symptoms, and their geographical distributions.

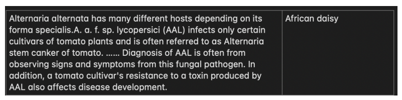
By analyzing this data, researchers can gain valuable insights into the spread and prevalence of plant diseases, as well as their potential impact on agricultural productivity and food security. Additionally, with the help of these NLP techniques, plant disease identification can be made accessible to individuals who may not have a deep understanding of plant diseases or have access to specialized knowledge or equipment.

To perform plant disease identification, we first scraped data from Wikipedia. Once we obtained the data, we augmented it using the GPT-3 API. We then pre-processed our data using the BERT tokenizer and trained the BERT model with multiple settings in order to achieve the best possible results. In the later sections of this report, we also compared the results of our text classifier with an image classifier model.

3 Data

The dataset used in the project is plant disease dataset which scraped from Wikipedia through Wikipedia API. We wrote a custom script to scrape data from Wikipedia API. And with this script we scraped around 1960 data points. Each data point comprises of a paragraph of the symptoms and the corresponding plant disease name.

To perform another experiment, we decided to combine the labels for diseases belonging to the same plant species. This was done in order to increase the number of examples in each class from just one example to at least 10-15 examples per class. However, in order for the model to generalize well to plant diseases specifically, rather than just plants in general, we performed data augmentation on the original dataset to yield more training examples per label.



Alternaria alternata has many different hosts depending on its forma specialis.A. a. f. sp. lycopersici (AAL) infects only certain cultivars of tomato plants and is often referred to as Alternaria stem canker of tomato. ...... Diagnosis of AAL is often from observing signs and symptoms from this fungal pathogen. In addition, a tomato cultivar's resistance to a toxin produced by AAL also affects disease development.	African daisy

Figure 1: Single Data Point

The entire dataset was devoted to training

Dataset Type	Size
Train	100% (1960)
Test	0% (0)
Validation	0% (0)

Table 1: Initial Dataset

dataset and none were allocated to validation and test datasets (see table 1. This data requires the model to perform 1-shot learning which leads to one the experiments.

4 Data Augmentation

As mentioned earlier, we were successfully in scraping the data from Wikipedia but the data had its own problem. For each unique label we had only 1 unique text. This means that we have 1960 different categories but for each category we have just one data point. So our aim was to resolve this problem by focusing on two major things: Either extract more data from different websites or use text augmentation methods to create new data from existing data.

Various tools, including Scrappy, Wikipedia API, and Apache Nutch, can be utilized for extracting data from websites. When it comes to information on plant diseases, Wikipedia proved to have the most comprehensive data compared to other websites. Additionally, all the data was structured in a fixed format, with each plant disease having a dedicated paragraph on its signs and symptoms. However, other websites only had data on some plant diseases, not all. Scraping such websites may not always be feasible, and random scraping can yield thousands of URLs with varying content. Nonetheless, the challenge with such an approach is that the data obtained may lack context or indication of which plant disease it pertains to. Thus, to address these challenges, text augmentation methods were employed to expand the dataset.

Text augmentation methods can be used to generate new data by replacing tokens, adding new ones, translating or paraphrasing. Libraries such as Parrot, Paraphrase, and Textattack are available for this purpose. However, these libraries are typically better suited for single sentences rather than large datasets. To overcome this limitation, we employed the GPT-3 API to paraphrase existing content or create new content for each label. Another approach we tried was to split the initial text (split the text whenever an end of statement full-stop is encountered) into multiple data points to generate more data for each label. This helped to increase the dataset size by generating more data points for each unique label. As a result, the model was able to better learn the patterns and improve its performance.

Using this method we generated 6 different datasets:

1. The original dataset (1960 labels, each label with 1 data-point).

2. The paraphrased dataset (Original + Paraphrased Text). Each label with 2 data-points.

3. The new dataset (Original + new GPT Text). Each label with 2 data-points.

4. The true dataset (Original + paraphrased + new GPT Text). Each label with 3 data-points.

5. The split dataset (Original + Split Text). Each label with more than 10 data-points.

6. The mixed dataset (Original + paraphrased + new GPT Split Text). Each label with more than 10 data-points (32.5K records)

5 Methodology

1. **Tokenizer** - The dataset created by the crawler will have the labels and the paragraphs associated with it. This step will involve converting the sentences of the paragraph to tokens which can be processed by the NLP models. For the purpose of our model, we used BERT Tokenizer which converts the input to an input format which can be processed by BERT model. We also used XLNet tokenizer to train the XLNet model.

2. **Additional Pre-Processing** - The labels of the dataset were also one hot encoded since the BERT Model [3] would return a one hot encoded label matrix as the output for multi-label.

3. **BERT Model**- A general architecture which our work was following is shown in Fig 2.

   We pass the tokenized input to the BERT model, and use the BERT embeddings to train our Fully connected layers. We use pre-trained BERT model since the dataset is not that large and pre-training the model for transformers will be difficult. However, fine

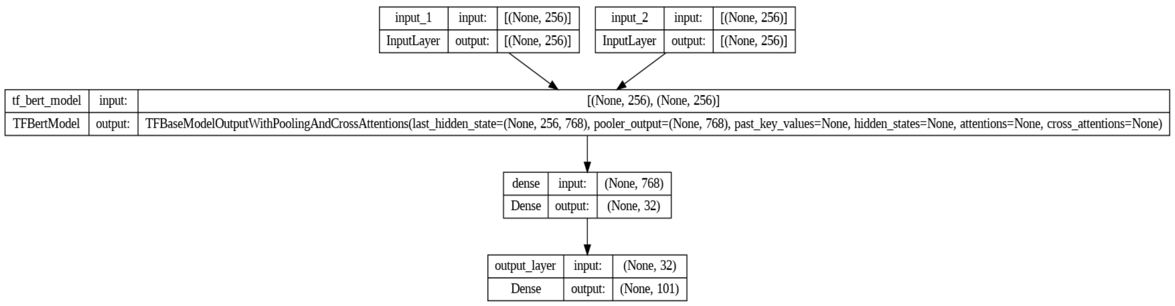

input_1 | input: | [(None, 256)]
InputLayer | output: | [(None, 256)]
input_2 | input: | [(None, 256)]
InputLayer | output: | [(None, 256)]
tf_bert_model | input: | [(None, 256), (None, 256)]
TFBertModel | output: | TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 256, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)
dense | input: | (None, 768)
Dense | output: | (None, 32)
output_layer | input: | (None, 32)
Dense | output: | (None, 101)

Figure 2: General BERT Framework

tuning is possible and necessary to train the model for a particular dataset. We fine tune the fully connected layers to fit the plant disease dataset.

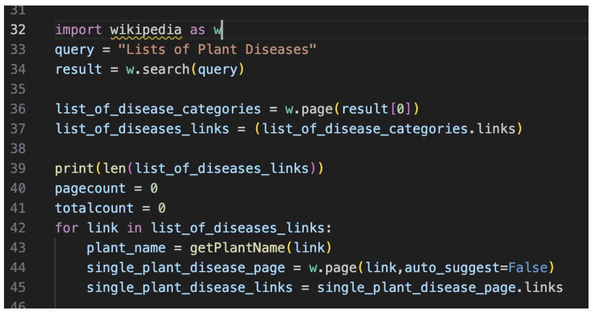
In the context of BERT [3], the central component is known as the Transformer, which incorporates self-attention as a crucial element. This relationship is exemplified in the following equation.

$$Attention\,(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

4. **CNN Model-** The research employed a deep convolutional neural network (CNN) to carry out plant disease identification. Prior to feeding the input images to the CNN, preprocessing was performed, and wavelet analysis was utilized to extract features, which ultimately reduced the number of input parameters. The identification of plant diseases was accomplished through transfer learning of the Inception Resnet model [6], which had been trained on a sufficiently large dataset. Unlike the Resnet model, the Inception Resnet model incorporates a modified module in conjunction with a stem module. The utilization of residual connections in the Inception Resnet model addresses the problem of degradation [4], while also reducing training time.

Learning Objective- The learning objective of the model was to see the interaction of different data sets with various settings of BERT model. Furthermore, we also want to see the comparative performance of image based models.

Inference Process of Errors- Based on the accuracies on the training and validation dataset, we infer if the trained model is overfitting or underfitting the data. We also try to infer the class imbalance of the dataset with the help of error and accuracy metrics.



```
import wikipedia as w
query = "Lists of Plant Diseases"
result = w.search(query)

list_of_disease_categories = w.page(result[0])
list_of_diseases_links = (list_of_disease_categories.links)

print(len(list_of_diseases_links))
pagecount = 0
totalcount = 0
for link in list_of_diseases_links:
    plant_name = getPlantName(link)
    single_plant_disease_page = w.page(link,auto_suggest=False)
    single_plant_disease_links = single_plant_disease_page.links
```

Figure 3: Data Scraping code snippet

6 Implementations

6.1 Libraries and Packages

- The study used many different packages which include Pandas, PyTorch, TensorFlow, Transformers (huggingface) and sklearn. Apart from these libraries we also used ”openai” for data augmentation and ”wikipedia” for data scraping.

6.2 Data Scraping

The figure 3 scrapes the data from wikipedia API which is limited to 5000 requests per hour. First there is a search query for the list of plant diseases and the all the webpages matching this search are returned. Next, we open the relevant page and get all the plant disease links. Once all the links are obtained, we extract data from each of the links/pages by visiting them programmatically.

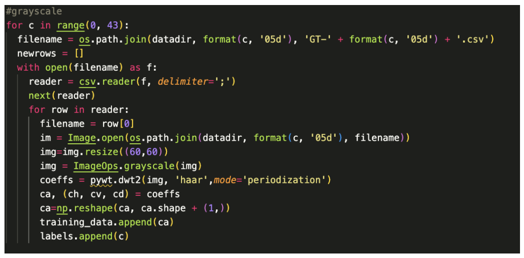
6.3 Data Augmentation

We utilized the openai python library to interact with the chat GPT API. Among the various GPT model engines available, we opted to use ”text-davinci-003” to paraphrase the original dataset 5, while for generating new text for each plant disease, we used ”gpt-3.5-turbo” 4. Notably, the davinci model outperforms other models in the chat GPT-3.5, offering faster processing speeds. However, it comes with a significant cost, charging 0.02/1k tokens. In contrast, the gpt-3.5-turbo is slower and has a limitation of only three requests per minute, but it is a more affordable model, charging just 0.002/1k tokens, which is only a fraction of the cost of the davinci model.

6.4 BERT

The figure 7 shows the code snippet which was used to train the main model. We tweaked the number of fully connected layers by adding hidden layers after bert embeddings. The number of nodes and layers will be determined after hyper-parameter tuning. Similarly, the optimizer and learning rates can also be tweaked to find the perfect fit. The various settings and results will be discussed in Section 7.

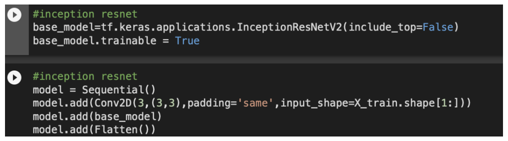
6.5 CNN



```python
#grayscale
for c in range(0, 43):
  filename = os.path.join(datadir, format(c, '05d'), 'GT-' + format(c, '05d') + '.csv')
  newrows = []
  with open(filename) as f:
    reader = csv.reader(f, delimiter=';')
    next(reader)
    for row in reader:
      filename = row[0]
      im = Image.open(os.path.join(datadir, format(c, '05d'), filename))
      img=img.resize((60,60))
      img = ImageOps.grayscale(img)
      coeffs = pywt.dwt2(img, 'haar',mode='periodization')
      ca, (ch, cv, cd) = coeffs
      ca=np.reshape(ca, ca.shape + (1,))
      training_data.append(ca)
      labels.append(c)
```

Figure 8: CNN Pre-processing Code

The images which were there as the input have to pre-processed. Fig8 shows the code segment where we resize the image, convert the image into grayscale, and apply discrete wavelet transform (DWT). DWT is used for the reduction of input parameters from the images which helps to increase the accuracy of the CNN architecture. This also reduces the training time significantly.



```python
#inception resnet
base_model=tf.keras.applications.InceptionResNetV2(include_top=False)
base_model.trainable = True

#inception resnet
model = Sequential()
model.add(Conv2D(3,(3,3),padding='same',input_shape=X_train.shape[1:]))
model.add(base_model)
model.add(Flatten())
```

Figure 9: CNN Code Snippet

To utilise the inception-resnet [6] architecture, we download the pre-trained model, and add a fully connected layer on top of that. Since, the dataset had enough data, we trained the entire architecture including the inception-resnet layers using our dataset.

7 Experiments and Results

7.1 Learning_rate

The current study, experiments with learning rates. We have used three different learning rates as described in Table 2. For the purpose of finding the optimum learning rate we have fixed the dataset to be the plant name dataset. The reason why the 1e-4 didnt result in a good fit is because it couldn't find the global minima and in case of 1e-6 the model was stuck in an local optima.

Learning_Rate	Train_Accuracy	Val_Accuracy
1e-4	5.1%	0%
1e-5	28.58%	27.75%
1e-6	13.25%	11.50%

Table 2: Learning_Rate

7.2 Hidden Layer

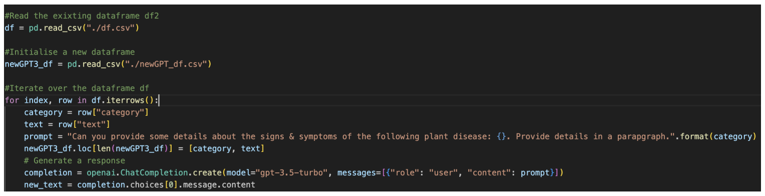
We have also experimented with different number of layers, and the findings are discussed in Table 3. We found out that the ideal number of layers to be 2. Similar to the learning_rate we fixed the dataset while tuning the number of hidden layers.

Hidden Layer	Train_Accuracy	Val_Accuracy
1	24.74%	24.25%
2	28.58%	27.75%
3	22.29%	23.75%

Table 3: Hidden Layer

7.3 Datasets

This was a very elaborate part of the study since we compared many different datasets as discussed

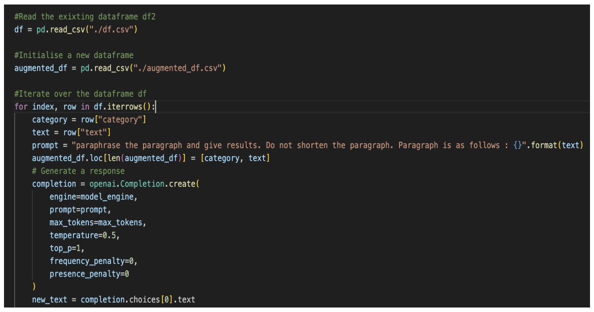


```python
#Read the exixting dataframe df2
df = pd.read_csv("./df.csv")

#Initialise a new dataframe
newGPT3_df = pd.read_csv("./newGPT_df.csv")

#Iterate over the dataframe df
for index, row in df.iterrows():
    category = row["category"]
    text = row["text"]
    prompt = "Can you provide some details about the signs & symptoms of the following plant disease: {}. Provide details in a parapgraph.".format(category)
    newGPT3_df.loc[len(newGPT3_df)] = [category, text]
    # Generate a response
    completion = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=[{"role": "user", "content": prompt}])
    new_text = completion.choices[0].message.content
```

Figure 4: Data Generation Through ChatGPT code snippet

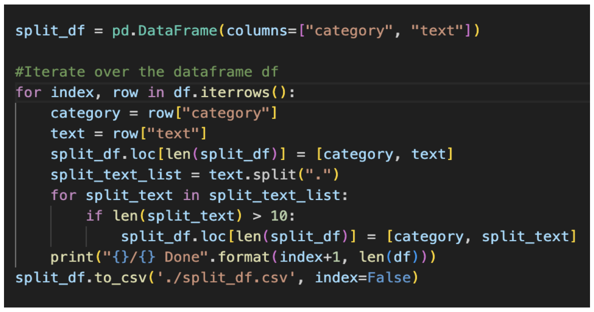


```python
#Read the exixting dataframe df2
df = pd.read_csv("./df.csv")

#Initialise a new dataframe
augmented_df = pd.read_csv("./augmented_df.csv")

#Iterate over the dataframe df
for index, row in df.iterrows():
    category = row["category"]
    text = row["text"]
    prompt = "paraphrase the paragraph and give results. Do not shorten the paragraph. Paragraph is as follows : {}".format(text)
    augmented_df.loc[len(augmented_df)] = [category, text]
    # Generate a response
    completion = openai.Completion.create(
        engine=model_engine,
        prompt=prompt,
        max_tokens=max_tokens,
        temperature=0.5,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    new_text = completion.choices[0].text
```

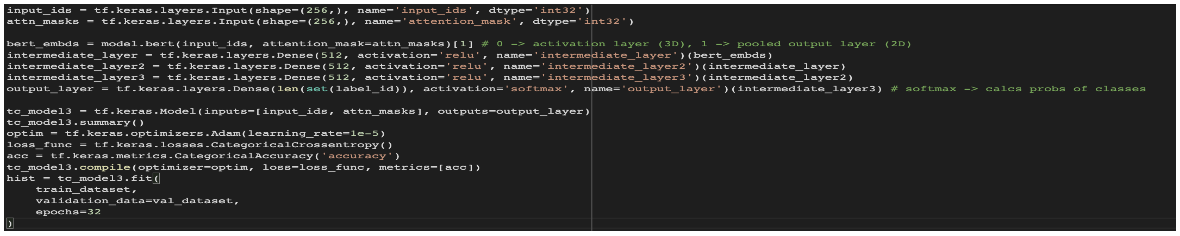
Figure 5: Data Paraphrasing Through ChatGPT code snippet



```python
split_df = pd.DataFrame(columns=["category", "text"])

#Iterate over the dataframe df
for index, row in df.iterrows():
    category = row["category"]
    text = row["text"]
    split_df.loc[len(split_df)] = [category, text]
    split_text_list = text.split(".")
    for split_text in split_text_list:
        if len(split_text) > 10:
            split_df.loc[len(split_df)] = [category, split_text]
    print("{}/{} Done".format(index+1, len(df)))
split_df.to_csv('./split_df.csv', index=False)
```

Figure 6: Splitting the original dataset code snippet



```python
input_ids = tf.keras.layers.Input(shape=(256,), name='input_ids', dtype='int32')
attn_masks = tf.keras.layers.Input(shape=(256,), name='attention_mask', dtype='int32')

bert_embds = model.bert(input_ids, attention_mask=attn_masks)[1] # 0 -> activation layer (3D), 1 -> pooled output layer (2D)
intermediate_layer = tf.keras.layers.Dense(512, activation='relu', name='intermediate_layer')(bert_embds)
intermediate_layer2 = tf.keras.layers.Dense(512, activation='relu', name='intermediate_layer2')(intermediate_layer)
intermediate_layer3 = tf.keras.layers.Dense(512, activation='relu', name='intermediate_layer3')(intermediate_layer2)
output_layer = tf.keras.layers.Dense(len(set(label_id)), activation='softmax', name='output_layer')(intermediate_layer3) # softmax -> calcs probs of classes

tc_model3 = tf.keras.Model(inputs=[input_ids, attn_masks], outputs=output_layer)
tc_model3.summary()
optim = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss_func = tf.keras.losses.CategoricalCrossentropy()
acc = tf.keras.metrics.CategoricalAccuracy('accuracy')
tc_model3.compile(optimizer=optim, loss=loss_func, metrics=[acc])
hist = tc_model3.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=32
)
```

Figure 7: BERT Code Snippet

in Section 4. The results are presented in the Figure 10.

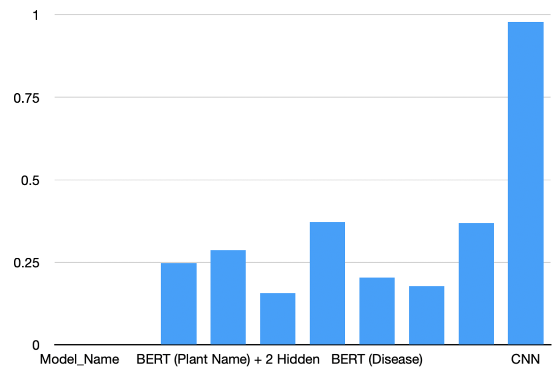


Figure 11: Training_Accuracy

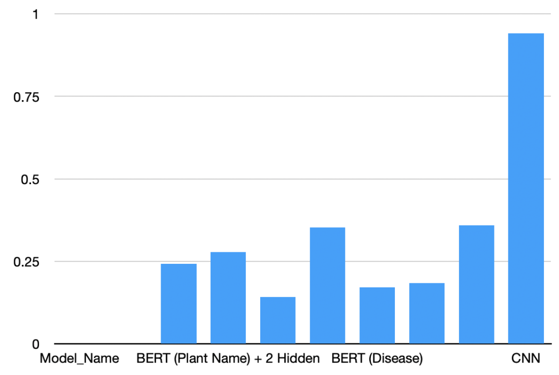


Figure 12: Val_Accuracy

Speed Analysis The models that we trained which included CNN and BERT based models. It was seen that a significant reduction of time was seen after using DWT as the feature extractor for CNNs. In case of BERT the training was very long going upto 4hrs for 32 epochs. This is mainly due to the hardware restrictions. However, the inference time of BERT models was slightly better the the CNN based models.

8 Conclusion

In our study, we found that the quality of data used to train a BERT model has a significant impact on its performance. Initially, we used 1-shot learning with only 1 data point per label, resulting in a mere 5% training accuracy. However, when we employed data augmentation techniques, our training and validation accuracy increased proportionally with the number of data points per label. Specifically, when we used 2 and 3 ”new” (not split) data points per unique plant disease label, our training reached 20%, and 37%, respectively 10. This demonstrates that our BERT model performs well when given more data to generalize across different categories.

Additionally, we compared our text classifier with an image classifier and found that the image classification model significantly outperformed the current text model. However, we strongly believe that this was due to the limited amount of training data for the text classifier, as opposed to the abundance of training data for the image classifier. As we aim to continue this study, it would be interesting to explore how a multi-model framework would perform when trained with both image and text datasets.

References

[1] Julian M. Alston and Philip G. Pardey. Agriculture in the global economy. *Journal of Economic Perspectives*, 28(1):121–46, February 2014.

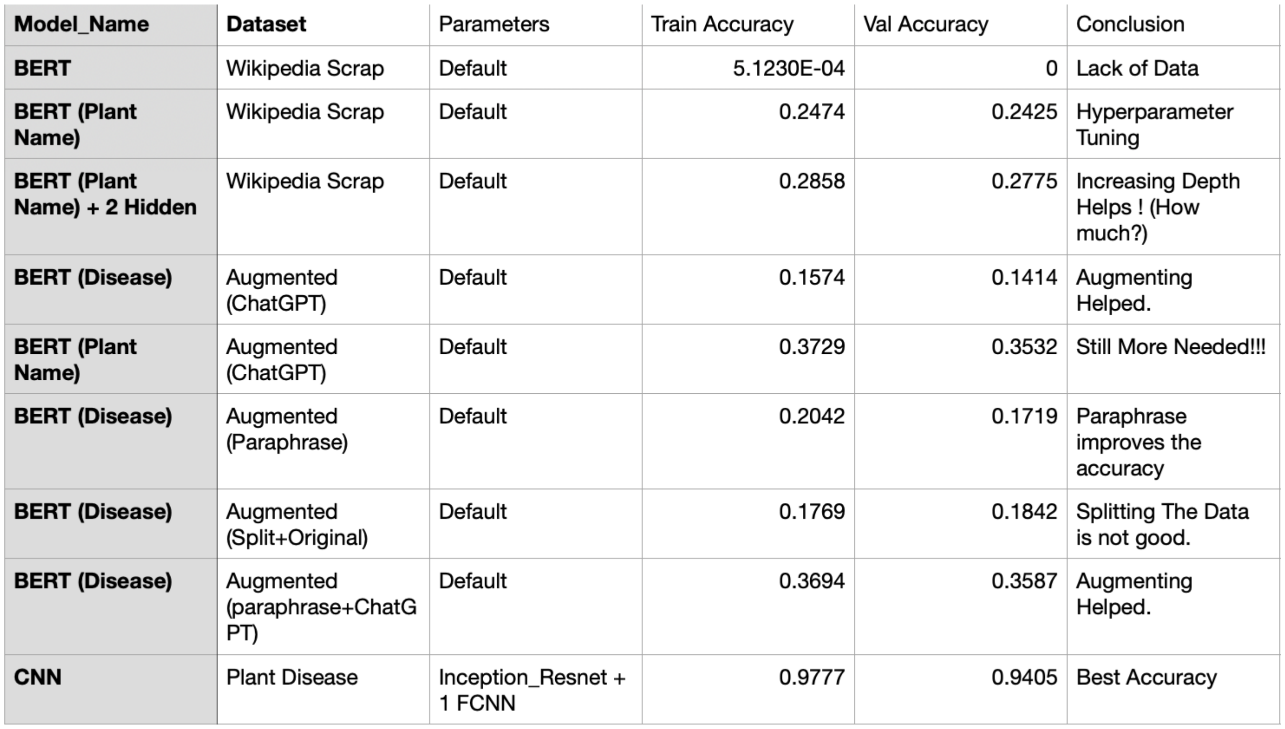
[2] Paul Christou and Richard M Twyman. The potential of genetically enhanced plants to address food insecurity. *Nutrition research reviews*, 17(1):23–42, 2004.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Long D Nguyen, Dongyun Lin, Zhiping Lin, and Jiuwen Cao. Deep cnns for microscopic image classification by exploiting transfer learning and feature concatenation. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

[5] Deeplata Sharma and DV Rao. A field study of pest of cauliflower cabbage and okra in some areas of jaipur. *International Journal of Life Sciences Biotechnology and Pharma Research*, 1(2):2250–3137, 2012.

[6] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.



Model_Name	Dataset	Parameters	Train Accuracy	Val Accuracy	Conclusion
BERT	Wikipedia Scrap	Default	5.1230E-04	0	Lack of Data
BERT (Plant Name)	Wikipedia Scrap	Default	0.2474	0.2425	Hyperparameter Tuning
BERT (Plant Name) + 2 Hidden	Wikipedia Scrap	Default	0.2858	0.2775	Increasing Depth Helps ! (How much?)
BERT (Disease)	Augmented (ChatGPT)	Default	0.1574	0.1414	Augmenting Helped.
BERT (Plant Name)	Augmented (ChatGPT)	Default	0.3729	0.3532	Still More Needed!!!
BERT (Disease)	Augmented (Paraphrase)	Default	0.2042	0.1719	Paraphrase improves the accuracy
BERT (Disease)	Augmented (Split+Original)	Default	0.1769	0.1842	Splitting The Data is not good.
BERT (Disease)	Augmented (paraphrase+ChatGPT)	Default	0.3694	0.3587	Augmenting Helped.
CNN	Plant Disease	Inception_Resnet + 1 FCNN	0.9777	0.9405	Best Accuracy

Figure 10: Final Results Table