Project Report: Leaf Species Classification

Dataset Description

The dataset consists of approximately 1,584 images of leaf specimens (16 samples each of 99 species) which have been converted to binary black leaves against white backgrounds. Three sets of features are also provided per image: a shape-contiguous descriptor, an interior texture histogram, and a fine-scale margin histogram. For each feature, a 64-attribute vector is given per leaf sample.

File descriptions

- **train.csv - the training set**
- **test.csv - the test set**
- **sample_submission.csv - a sample submission file in the correct format**
- **images - the image files (each image is named with its corresponding id)**

Data fields

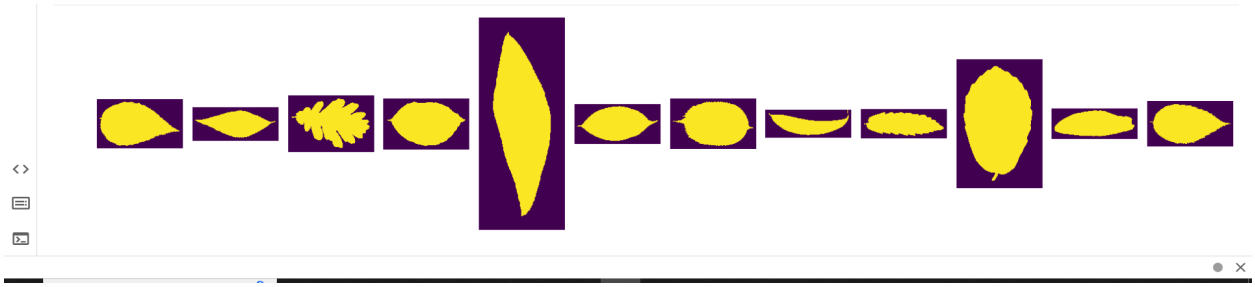
- **id - an anonymous id unique to an image**
- **margin_1, margin_2, margin_3, ..., margin_64 - each of the 64 attribute vectors for the margin feature**
- **shape_1, shape_2, shape_3, ..., shape_64 - each of the 64 attribute vectors for the shape feature**
- **texture_1, texture_2, texture_3, ..., texture_64 - each of the 64 attribute vectors for the texture feature**

Part I: Data Preparation

1. **importing the required libraries.**

```
[ ] #import libraries
    import pandas as pd
    import numpy as np
    import plotly.express as px
    import matplotlib.pyplot as plt
    from sklearn.decomposition import PCA
    from sklearn.manifold import TSNE
    import random
    from glob import glob
    import cv2
    from sklearn.preprocessing import LabelEncoder,StandardScaler
    from sklearn.model_selection import train_test_split
    import seaborn as sns
    from keras.models import Sequential
    from keras.layers import Input,Dense,Flatten,Dropout
    from tensorflow.keras.optimizers import SGD, Adam, RMSprop
    from tensorflow.keras import regularizers
    from tensorflow.keras.callbacks import EarlyStopping
    from keras.models import Sequential
    from keras.regularizers import l2
    import matplotlib.pyplot as plt
    from keras.utils import to_categorical
    import keras
```

2. **Loading the dataset**
3. **Describe the data set that provides insights into its characteristics**
4. **Clean the data set (Ensure data cleanliness by checking for null values and duplicates. By addressing these issues, the dataset can be rendered free of any nulls or duplicate entries, enhancing its reliability and suitability for analysis)**
5. **Data Visualization using PCA, T-SNE**
6. **Drawing some images**



7. **Carry out required correlation analysis(plot heatmap to drop the high correlated features**

8. **Checking the number of classes and finding them =99**

9. **Divide the data into data and target (`species`)**

10. **Data Standardization using mean and standard deviation**

11. **Encoding the labels using Label Encoder**

Part II: Training a neural network

1. **Splitting the data into training and test and the training data into training and validation**
2. **Build two functions to train and evaluate the model in different hyperparameters**

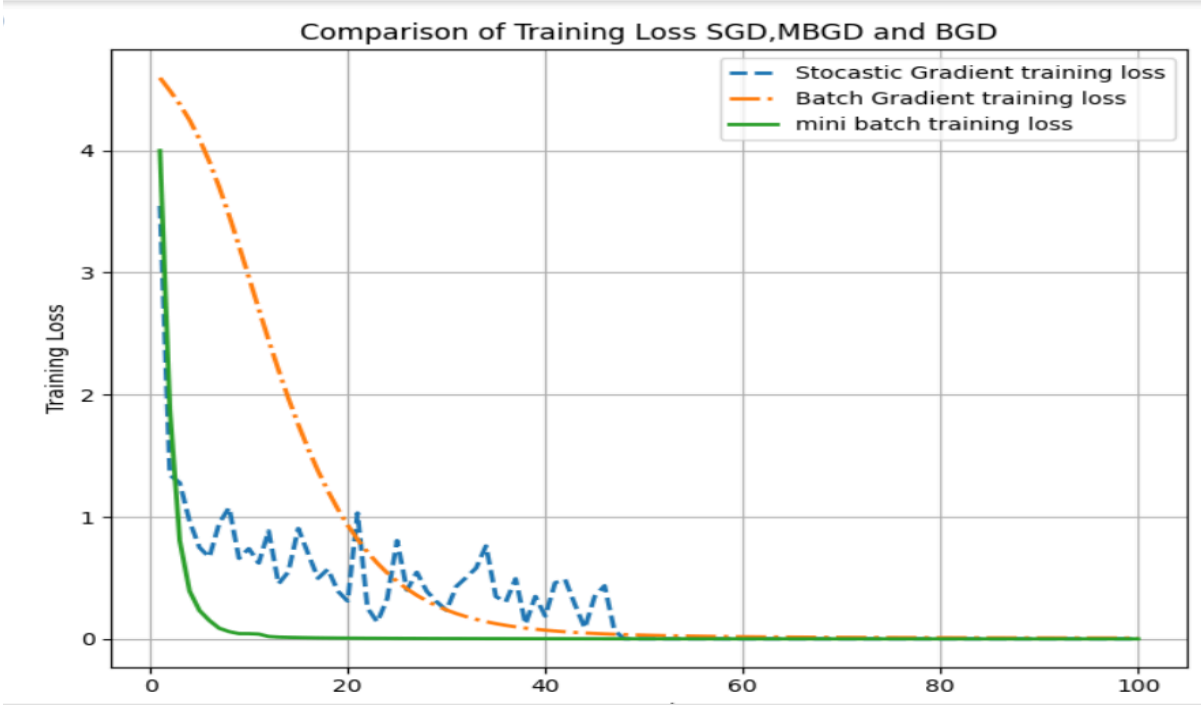
Hyperparameters

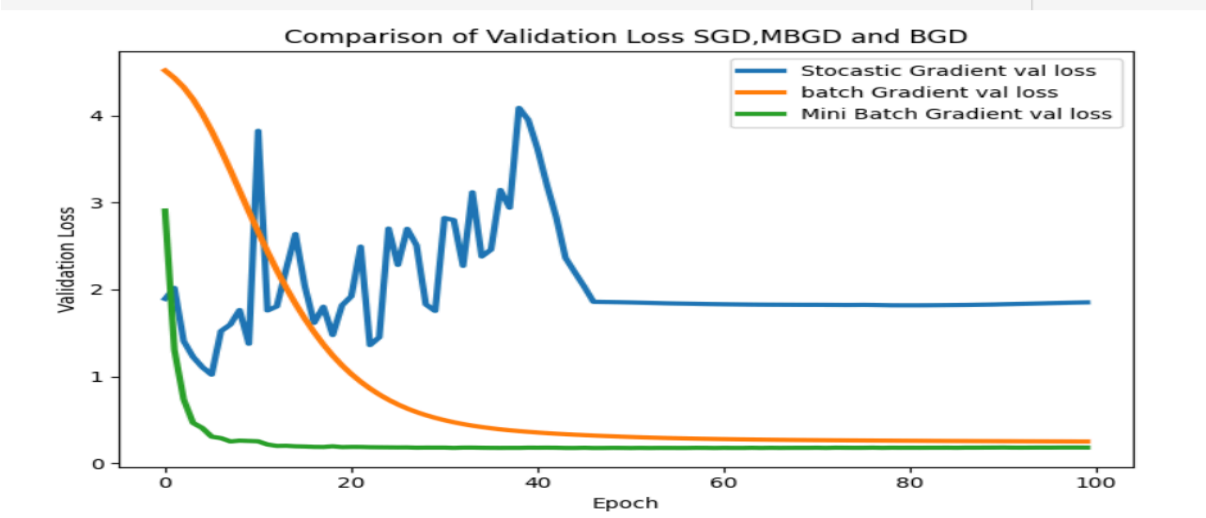
Tuning the Batch size:

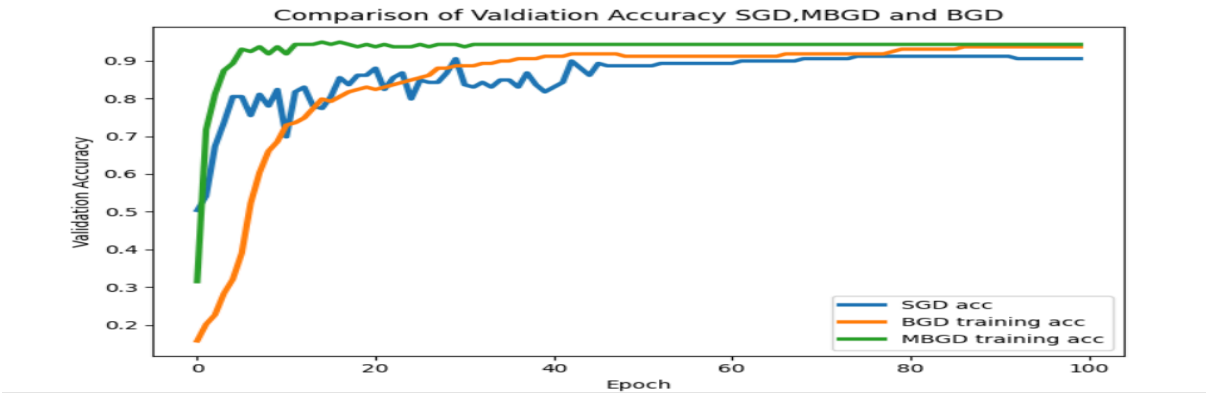
in this experiment I will try to compare the effect of changing the batch size by making the hyperparameters constant with this set of values hyperparamters= {'number_of_nodes':128, 'reg':0, 'epochs':100, 'optimizer':'adam', 'LR':0.1}

and observe the effect of choosing different Batch sizes:

- **I will start with Batch size =1 which means that I will Try SGD.**
- **then I will try Batch size = 633 which means True BGD.**
- **followed by different Batch Sizes as [16,32,64].**



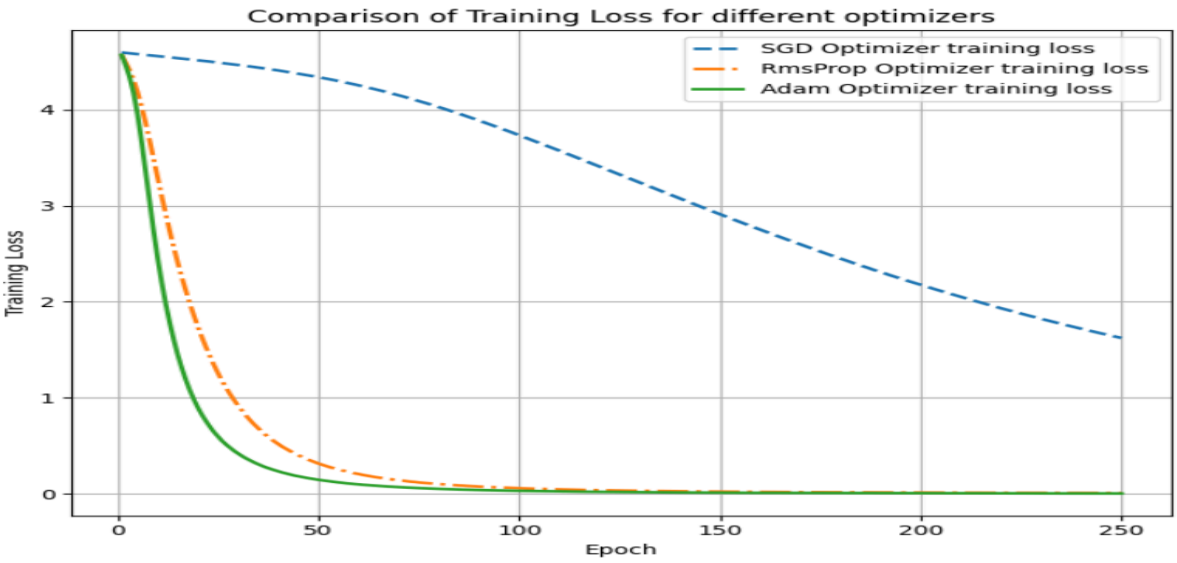


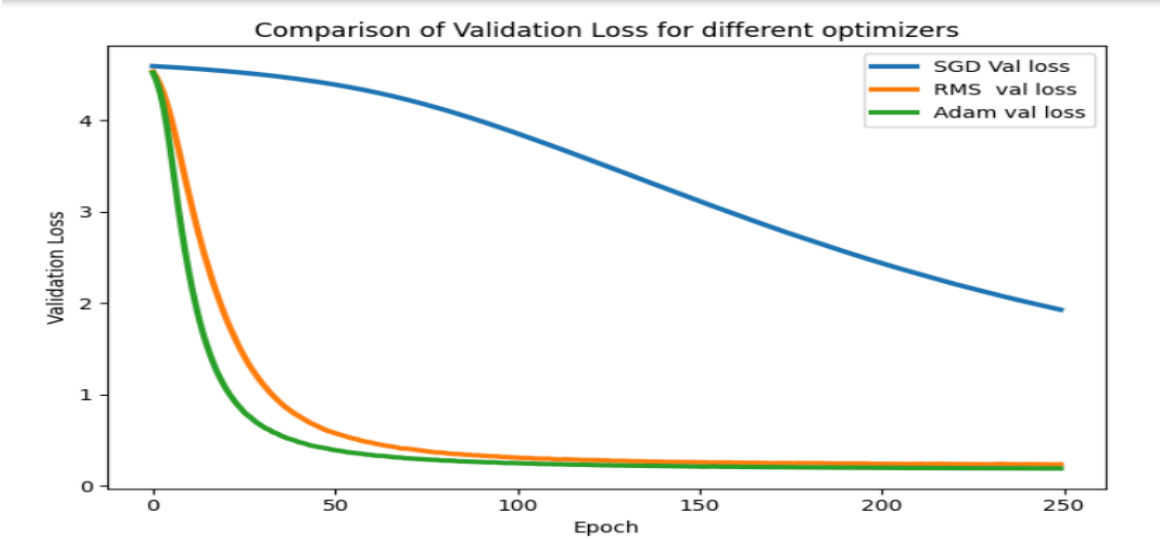


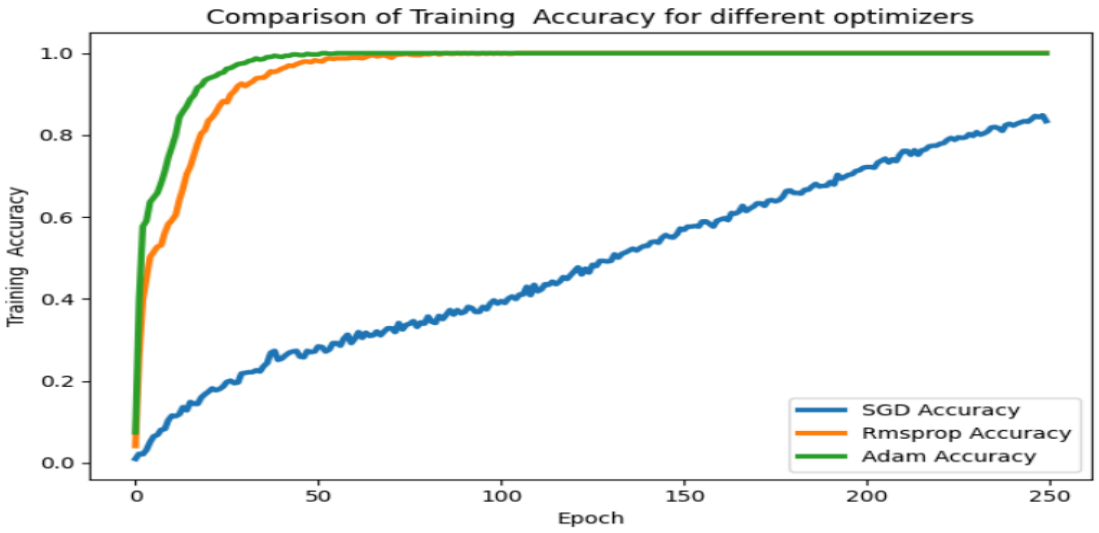
- **as we can see here from the Graphs The mini-batch Gradient Descent is the most suitable for our problem I will try to tune this in the next experiment after trying different optimizers, although through the training I also observed the training time for each model and as expected the SGD takes the most times this is because of the number of using the update rule in the backpropagation, compared to the BGD model which takes the least time in the computation as it only updates 100 times through the training which is the number of epochs, while the Mini batch gradient descent model was between them in the time comparison.**
  for the convergence as expected the SGD model contains oscillations in the loss curve, while the BGD model takes more epochs to converge but the curve was smoother and of course, the MBGD was in between both of them.

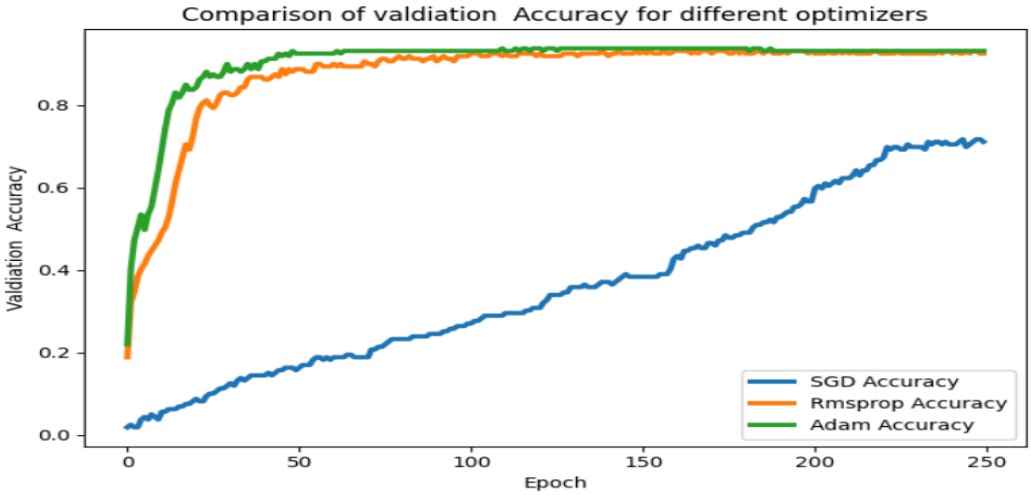
comparing different optimizers:

in this Experiment, i will try to compare different optimizers as SGD, momentum, and Adam with the following hyperparameters:hyperparameters={ 'number_of_nodes':512, 'reg':0, 'epochs':250, 'batch_size':32, 'LR':0.1, }

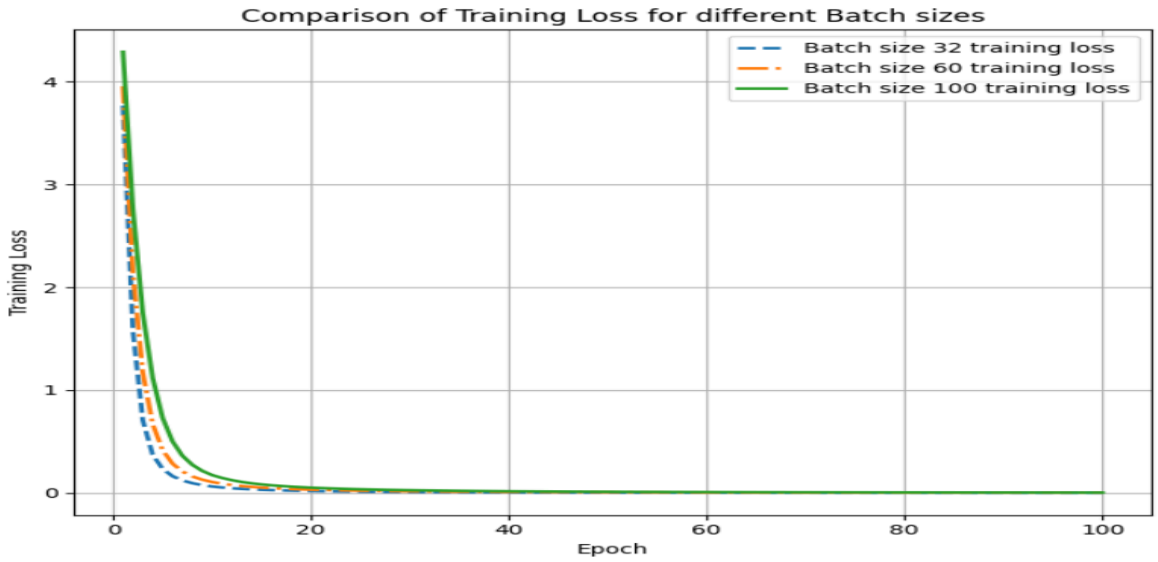


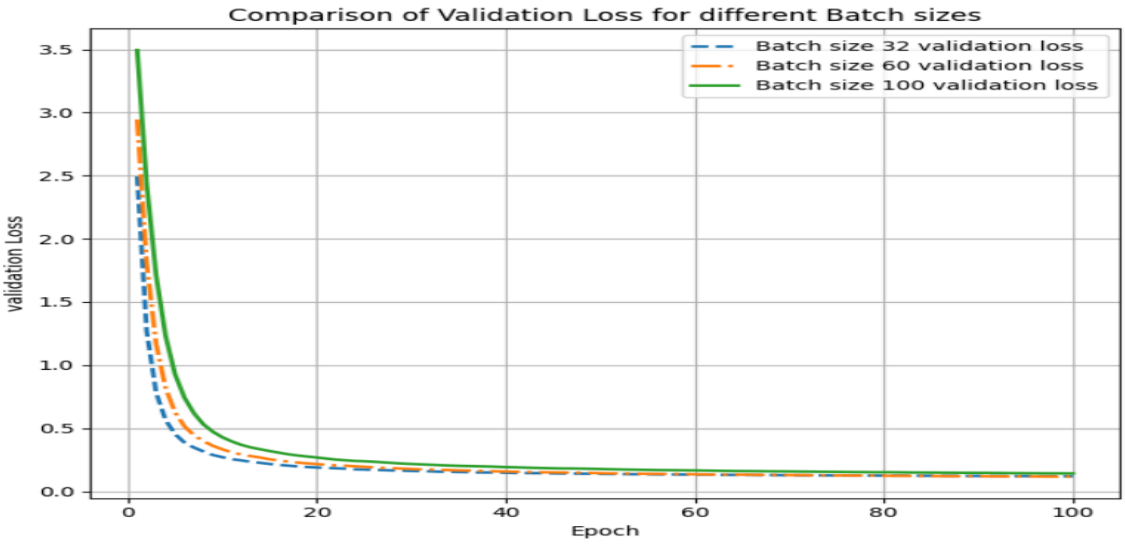


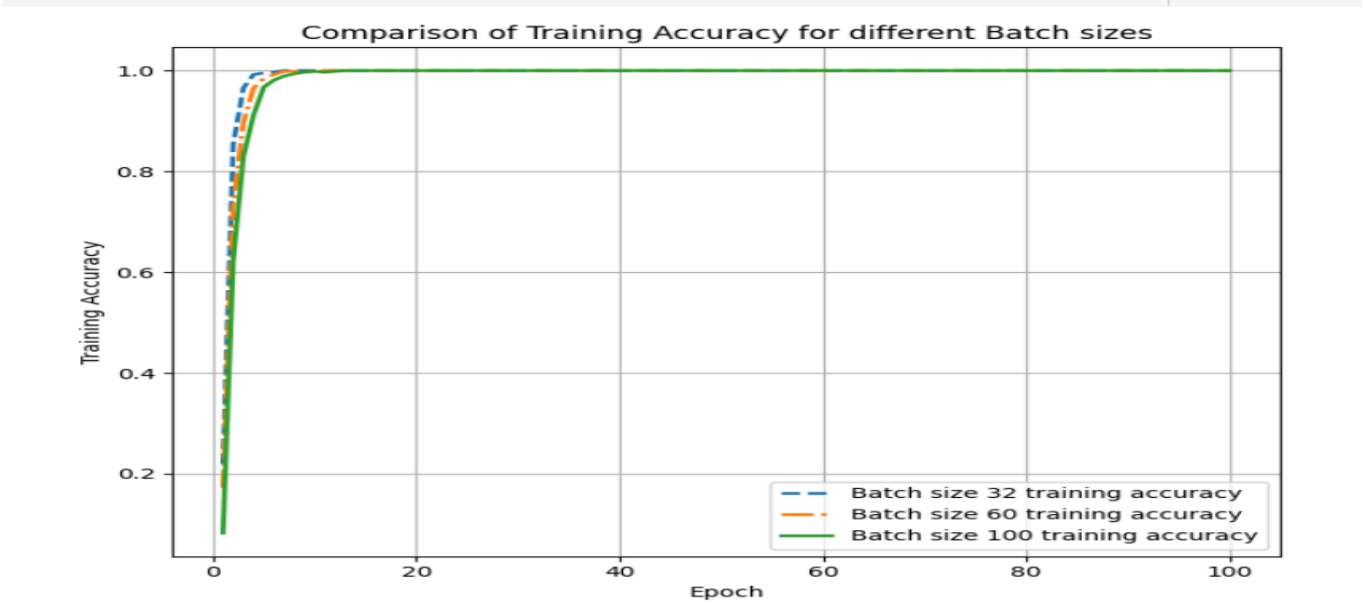


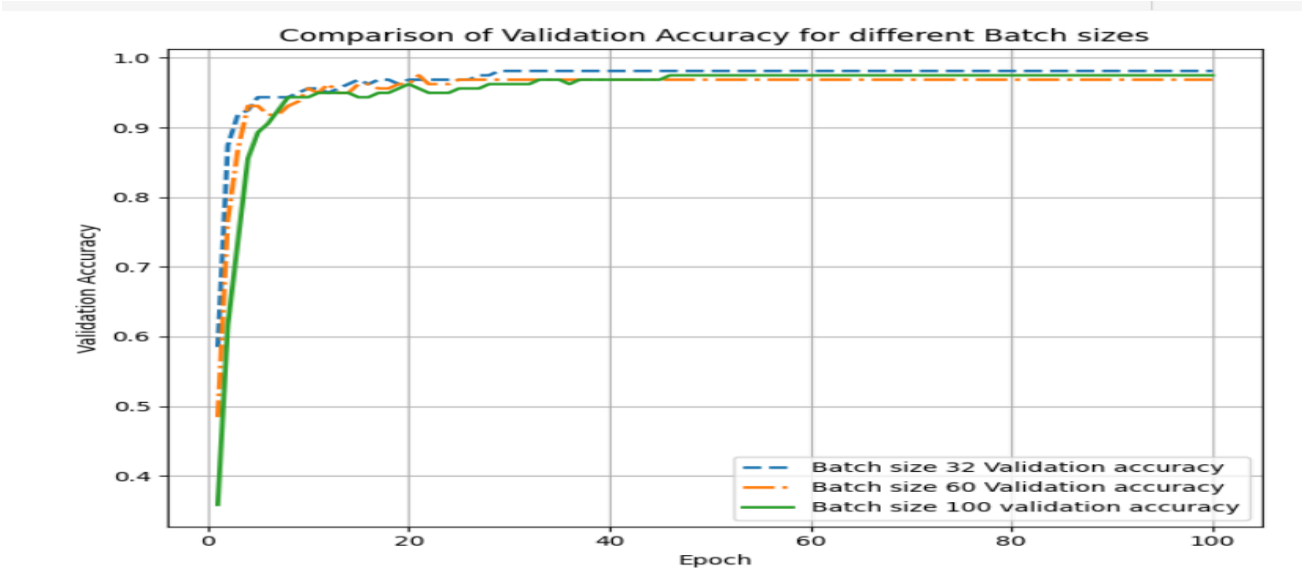


As we can see from the graphs both the Adam and Rmsporp optimizers could converge faster than the SGD optimizer, in the graphs I have observed that the SGD optimizer needs more epochs to converge. but I have decided to continue the project using ADAM optimizers.


Comparison of Training Loss for different Batch sizes
Batch size 32 training loss
Batch size 60 training loss
Batch size 100 training loss
Training Loss
Epoch
0
1
2
3
4
0
20
40
60
80
100


Comparison of Validation Loss for different Batch sizes
Batch size 32 validation loss
Batch size 60 validation loss
Batch size 100 validation loss
validation Loss
Epoch
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100



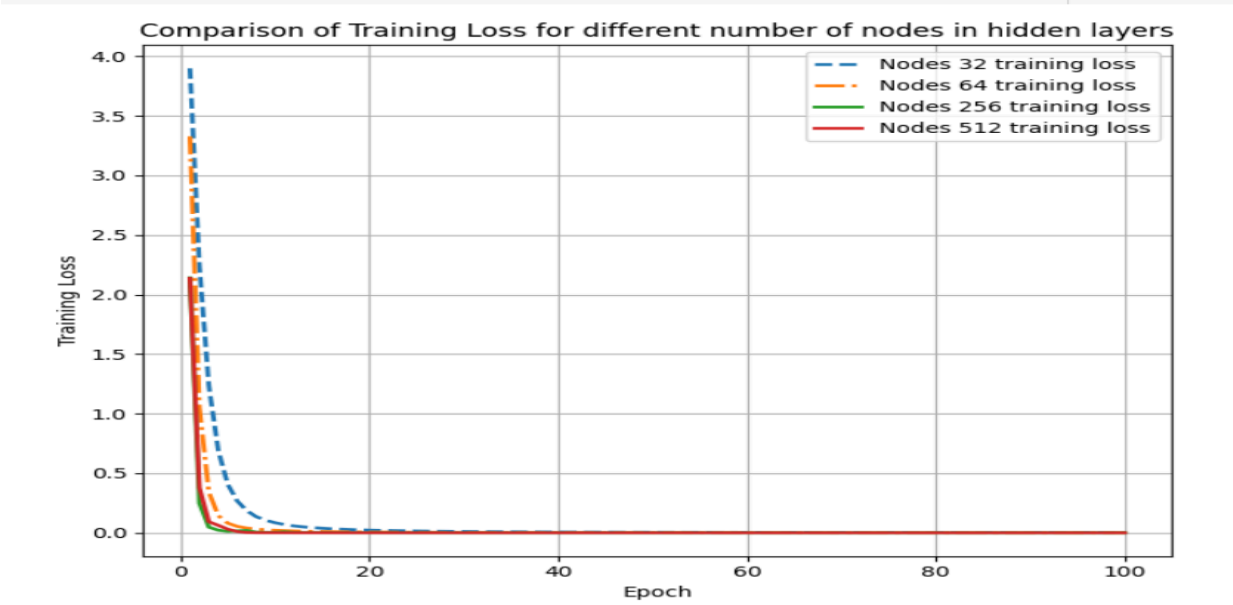


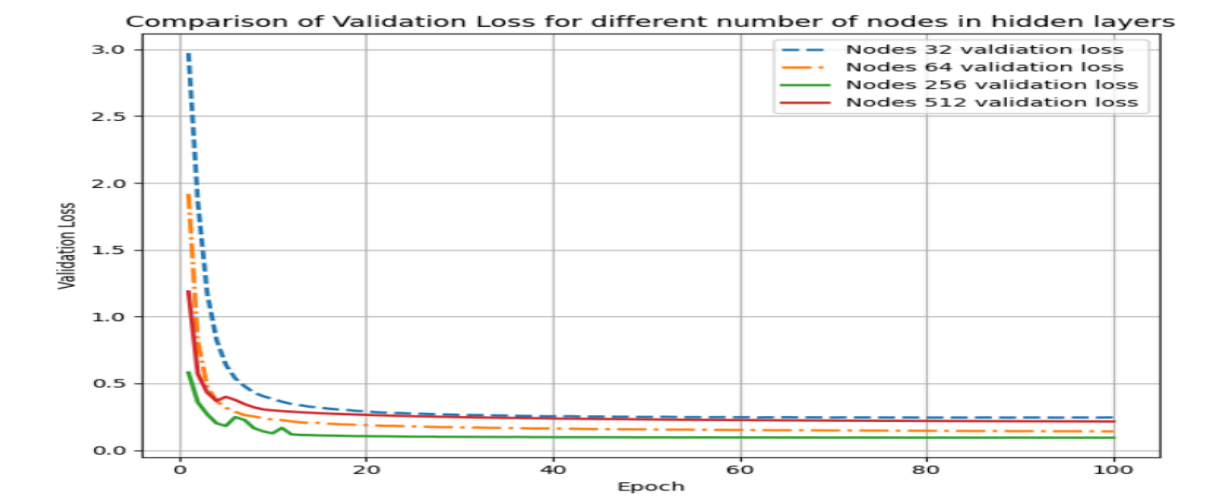
As we can see from the graphs as we increase the batch size we are getting closer to the BGD which makes the convergence slower.

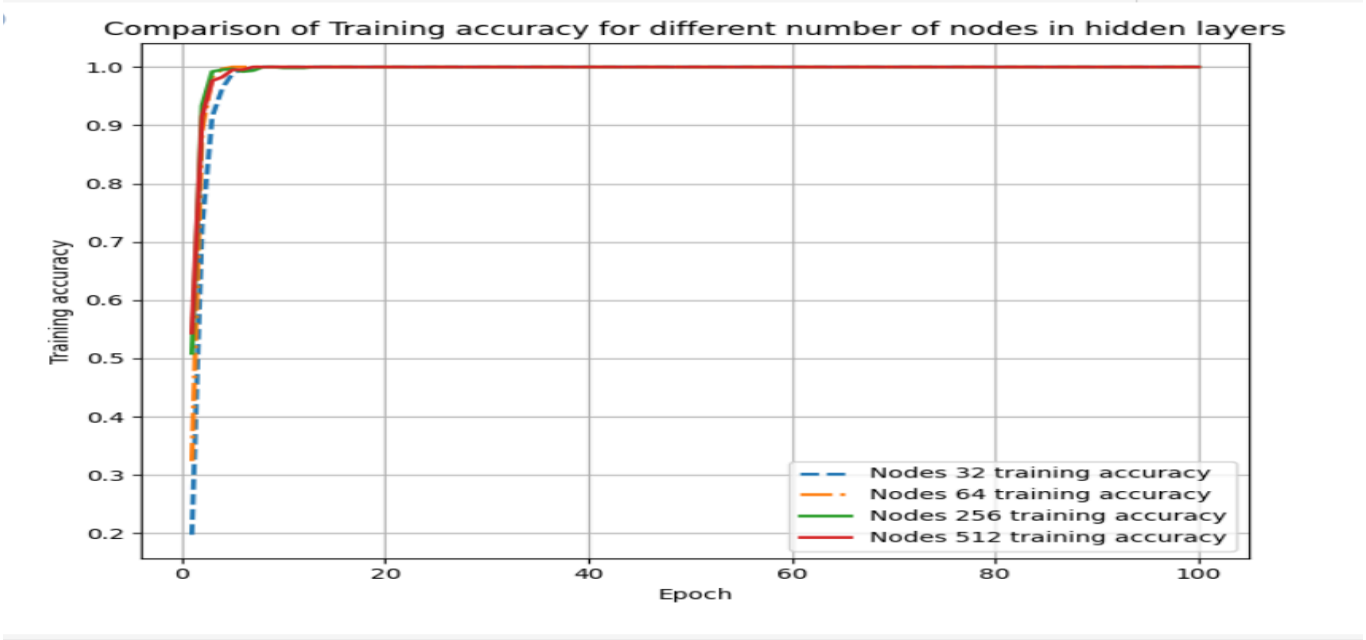
tune the number of nodes:

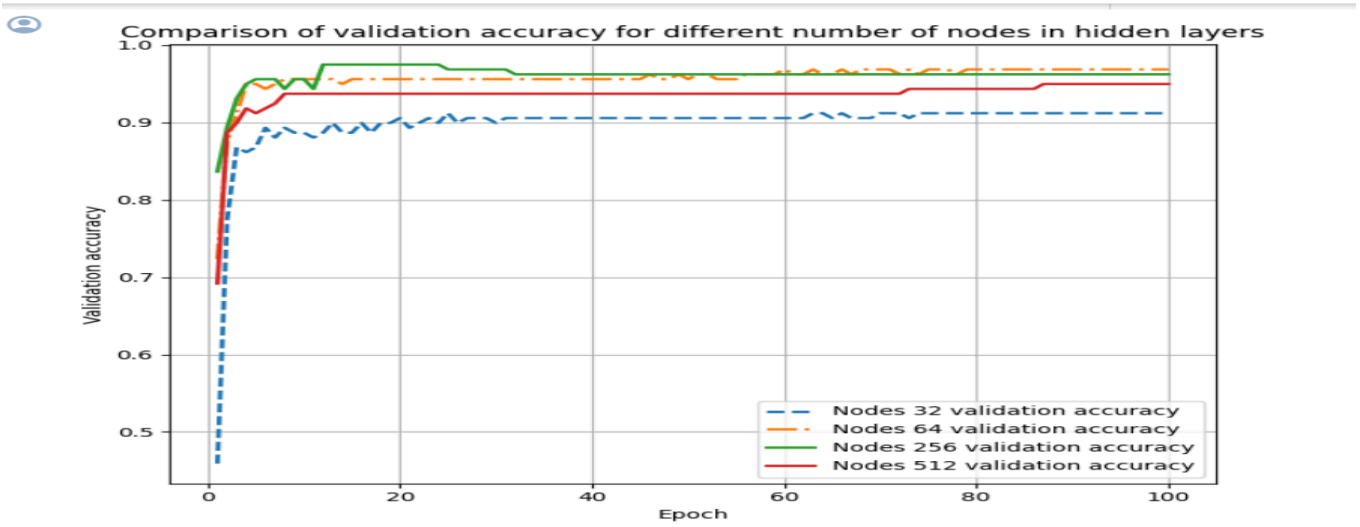
in this experiment, I will try different numbers of hidden nodes by making the other hyperparameters as below:

```
hyperparameters = {
   'reg': 0,
    'batch_size': 32,
    'epochs':100,
    'optimizer':'adam',
    'LR': 0.01,
}
```







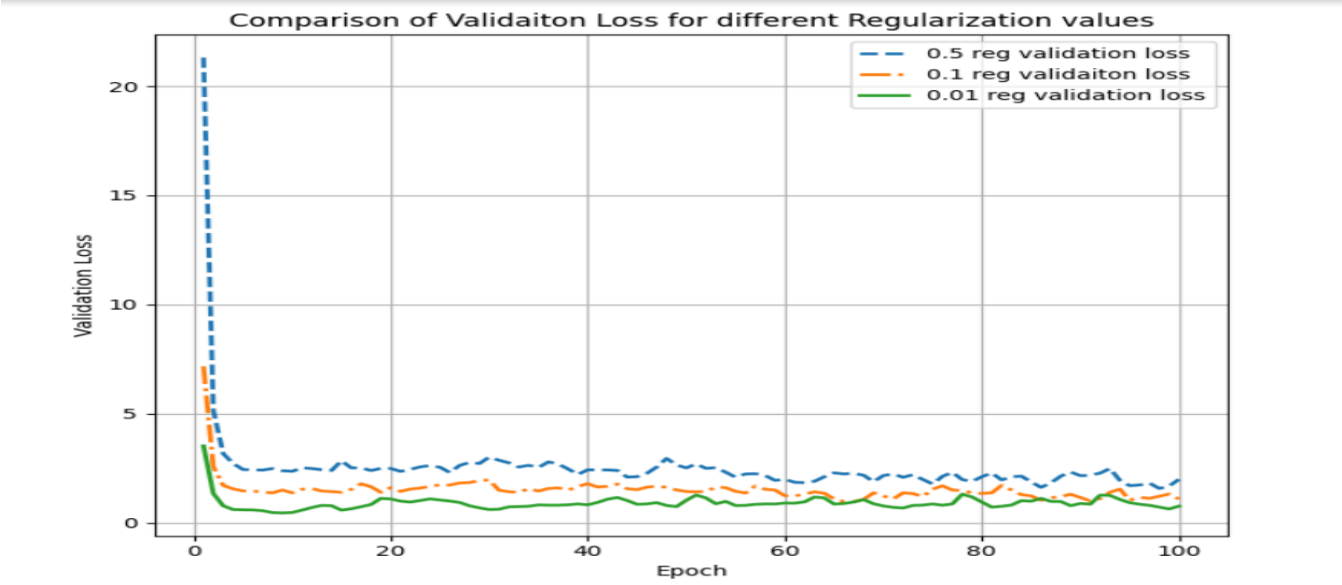


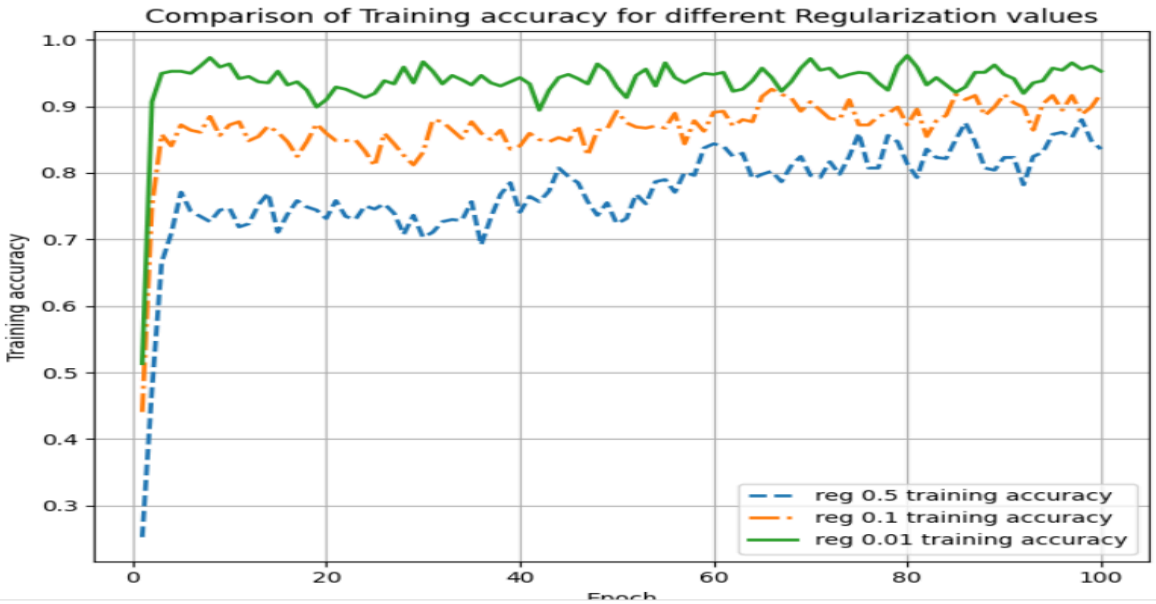
As we can see from the graphs increasing the number of nodes in the hidden layer helps the model to capture more complex patterns in the data.

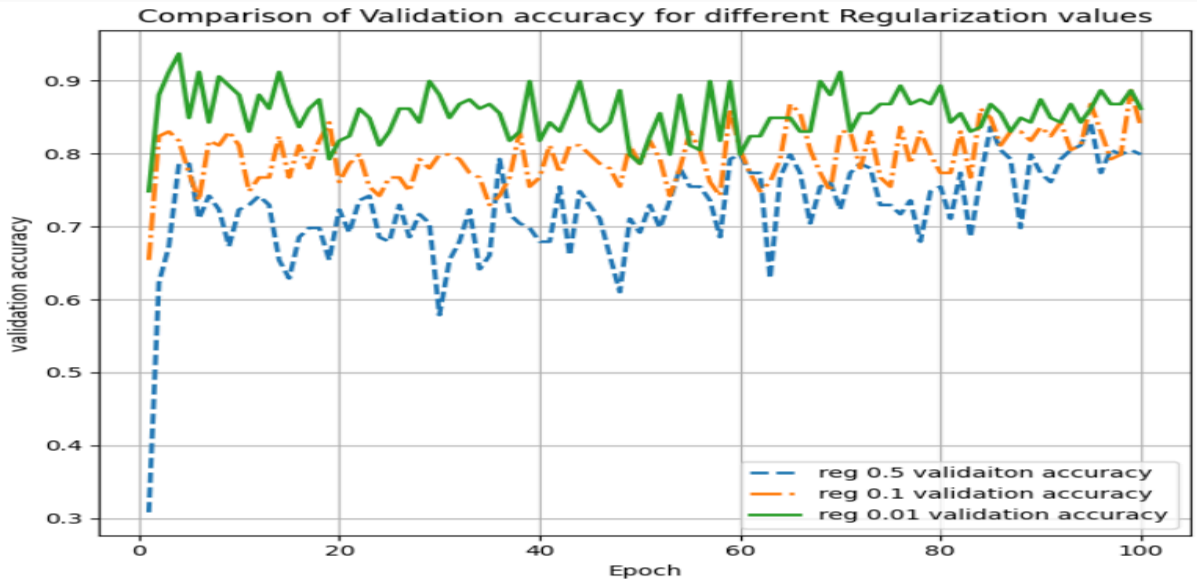
for the model with 256 hidden nodes I can see the evaluation on the train data was 100% accurate while in the validation data, it was 0.95 so in the upcoming experiments, I will try to use both the regularization and dropout to make those values close to each other.

Tuning the Regularization:

i will try the following values for regularizaion[0.1,0.01,0.001] with the model with those hyperparameters which i get from the previous experiment: hyperparameters = { 'number_of_nodes':256, 'batch_size': 32, 'epochs': 100, 'optimizer':'adam', 'LR': 0.01, }





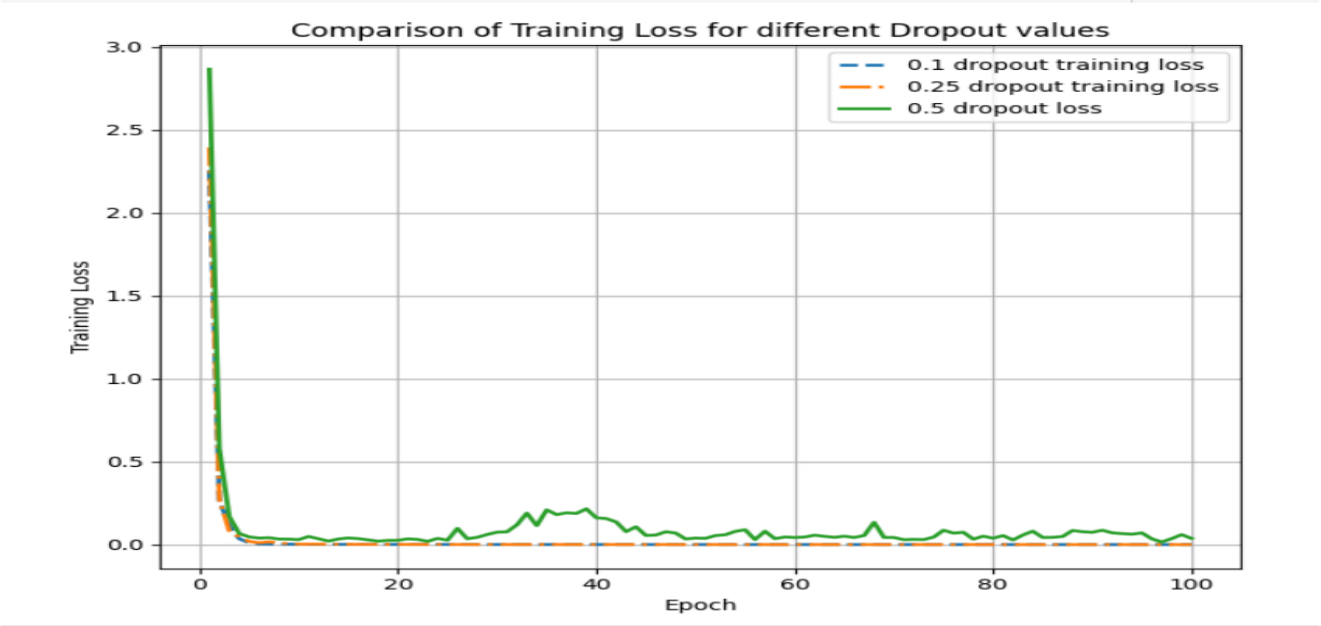


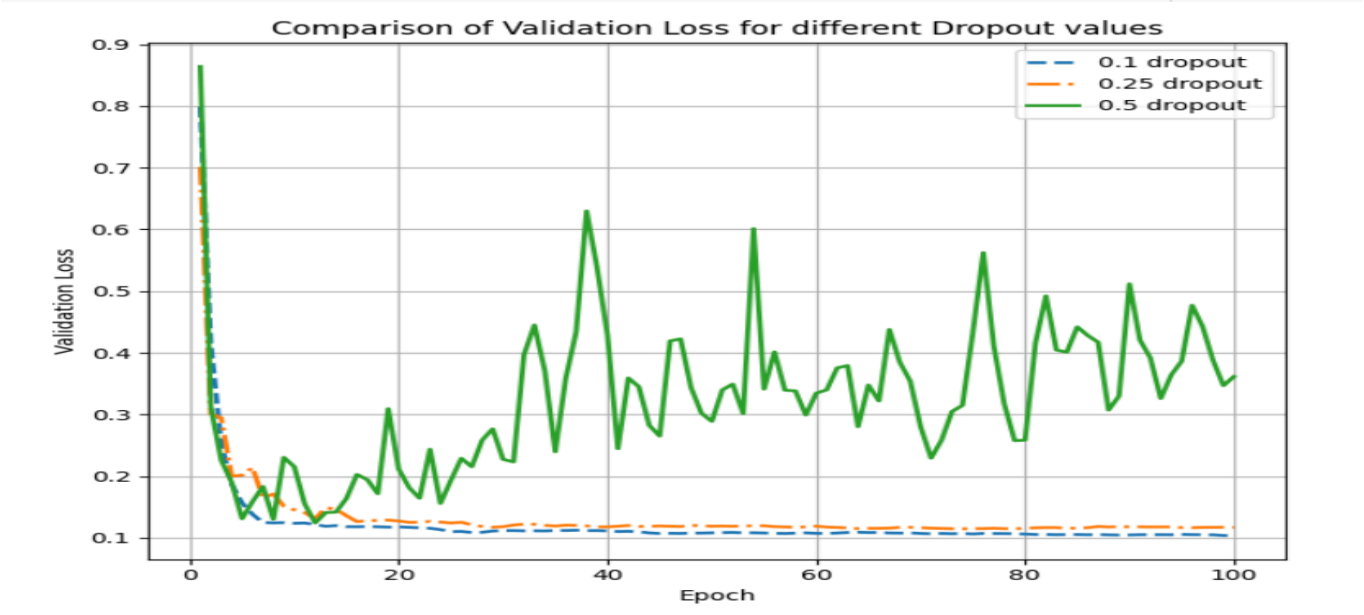
conclusion:

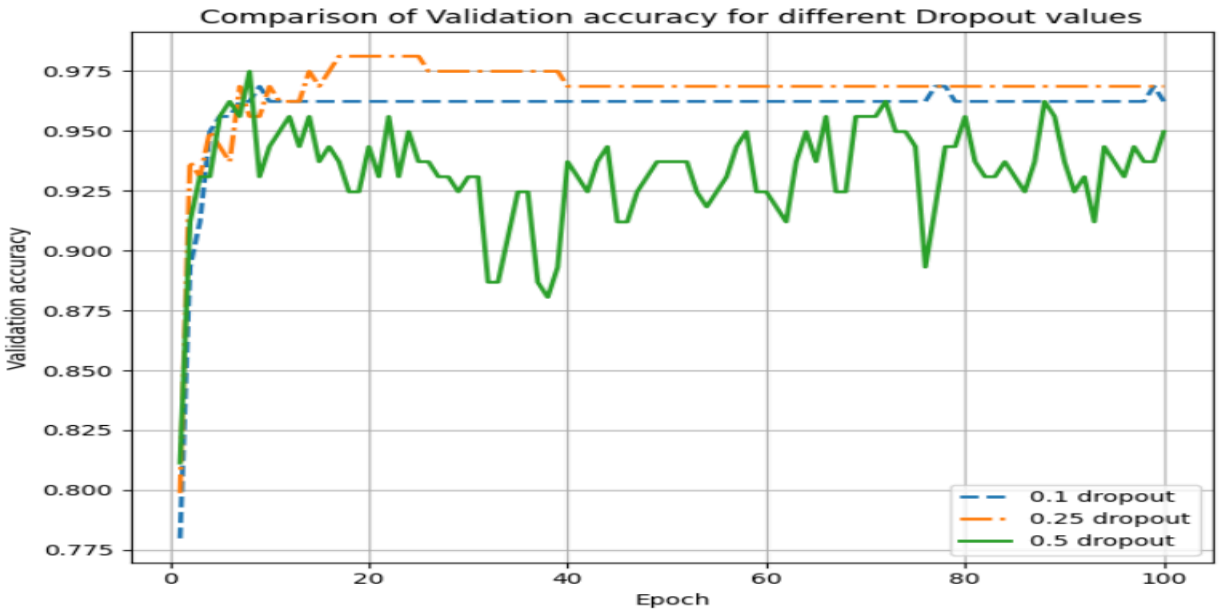
as expected increasing the regularization value, would affect the model accuracy in capturing the patterns in the data, as we can see from the above graphs the regularization term with value =0.5 was really bad in the evaluation on both the train and validation sets. so to try to reduce the gap between the train and validation accuracy i will try the dropout technique in the next experiment and i will set the regularization to zero again as it didn't help.

Tuning the dropout Ratio:

i will try the following values for drop-out ratio [0.1,0.25,0.5] with the model with those hyperparameters which i get from the previous experiment: hyperparameters = { 'number_of_nodes':256, 'batch_size': 32, 'epochs': 100, 'optimizer':'adam', 'LR': 0.01, 'reg':0 }







conclusion:

as we can see from the above graph the dropout could really help to reduce the gap between the accuracy on the training and validation sets, but also as we increase the value of the dropout ratio to 0.5 this leads to worse stability and accuracy. so after those experiments, i have found that the model with the following hyperparameter

{'number_of_nodes':264, 'reg': 0, 'batch_size': 32, 'epochs': 100, 'optimizer':'adam', 'LR': 0.01, 'dropOut':0.1 }

is the best model to evaluate on the test data,

Final evaluation:

after getting the best hyperparameter in the next cells i will try to evaluate the model with the above hyperparameters.