Queens’ University

Deep Learning

CISC 867 Project_1: Leaf Classification

Supervised by:

Prof. Hazem Abbas

T.A. Ahmed Nagib

Team members:

Sara Ahmed Mohamed Elfetiany

Mahmoud Mohamed Abdelhamied Abdin

Mostafa Mohamed Abdelhamid Azazi

CONTENTS

1. Introduction

There are estimated to be nearly half a million species of plant in the world. Classification of species has been historically problematic and often results in duplicate identifications.

In order to solve this issue, we will use the Leaf Classification dataset from Kaggle.com site and apply our Neural Network model on it with different Hyperparameters

So, Our objectives represent in accurately identify the 99 species of plants that the dataset contains using the different neural network hyperparameters

2. Data 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.

3. Data Preprocessing

3.1. Check the missing data

The first step is to check if there is any missing data in the dataset to clean it and replace it with appropriate value.
And by checking our dataset we didn’t find any missing data.

3.2. Check duplicates:

No duplicates found.

3.3. Encoding the data

The only column that contains categorical data and needs to be encoded is our target column ‘`species`’.

3.4. Check if there are outliers

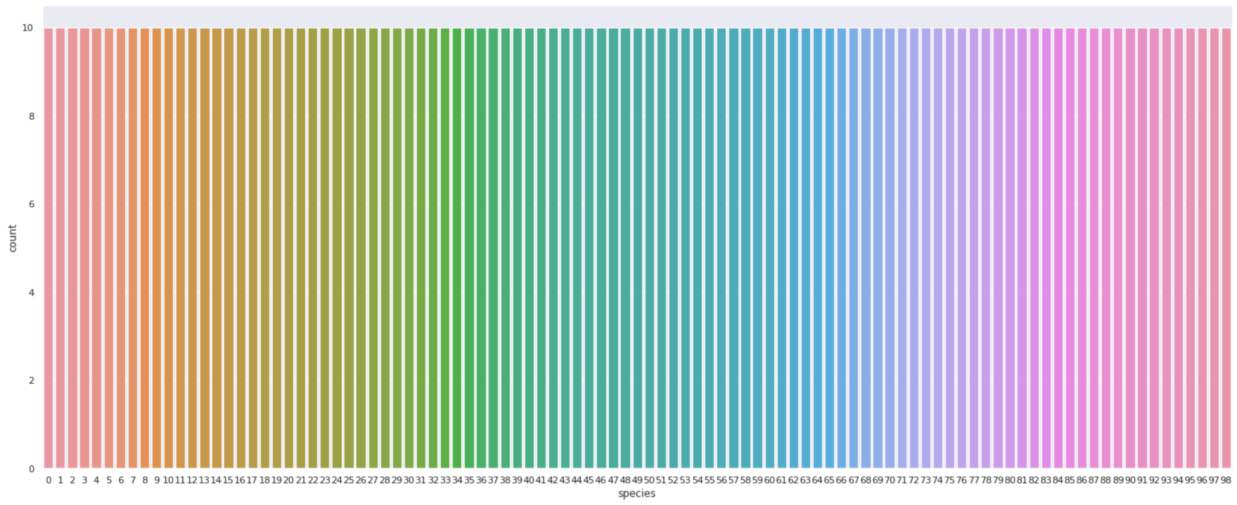
We checked the outliers for each class and no outliers were found.
We used z-score which is a numerical measurement that describes a value's relationship to the mean of a group of values.
And took a threshold of 3 after taking the abs value as any z-score greater than 3 or less than -3 is considered outlier.

4. Data visualization

4.1. Class counts and histograms of features

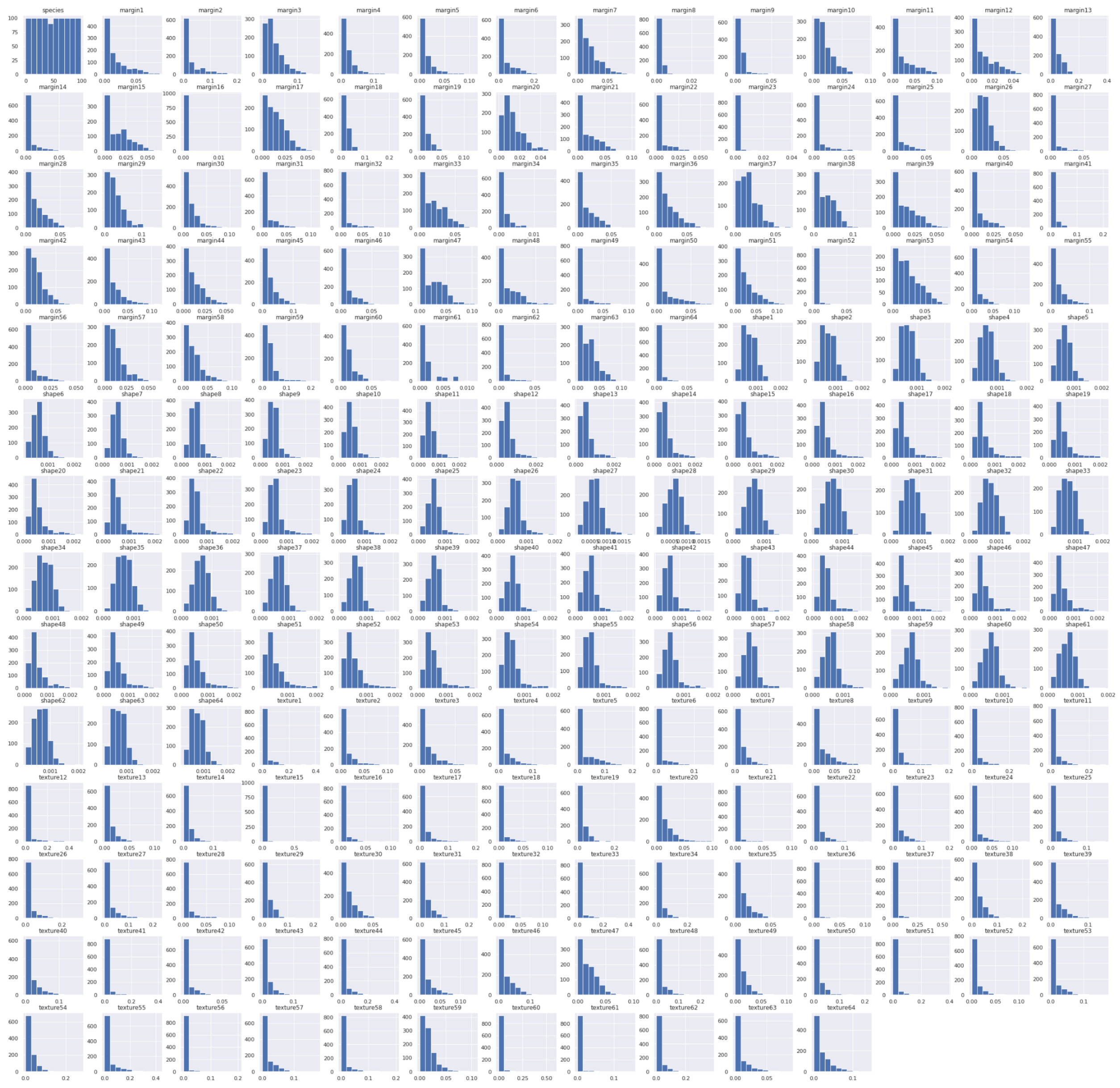
4.1.1. Spices Count

Plotting the count of each spices and we can see that the dataset is balanced as the data contain the same number of data points for each target category.



4.1.2. Data Histogram

From histogram we noticed that shape features are normally distributed
But the margin and texture features are right skewed.

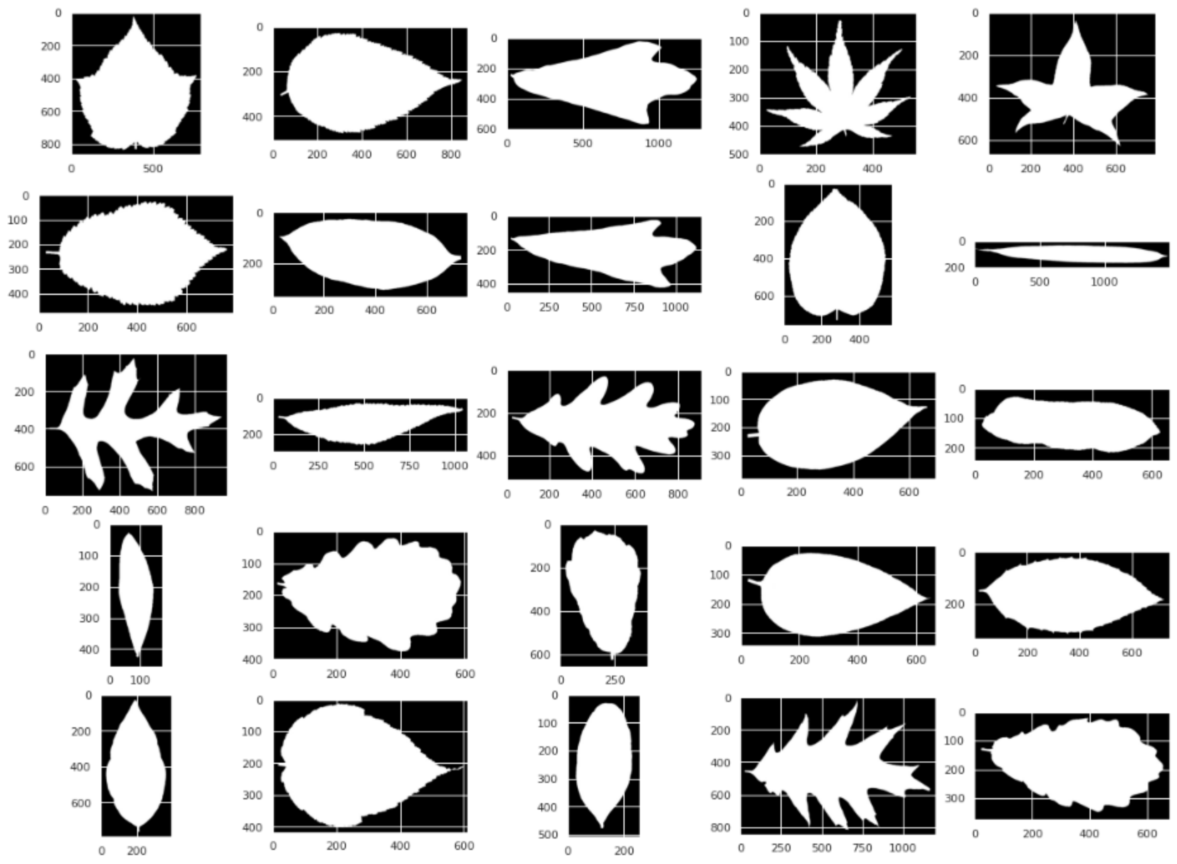


4.2. Random sample from the data

Plotting some random image from our dataset

```
import os
from keras.preprocessing.image import load_img
plt.figure(figsize=(20,15))

for i in range(25):
    j=np.random.choice((os.listdir('images')))
    plt.subplot(5,5,i+1)
    img=load_img(os.path.join('images',j))
    plt.imshow(img)
```



4.3. Correlation Analysis

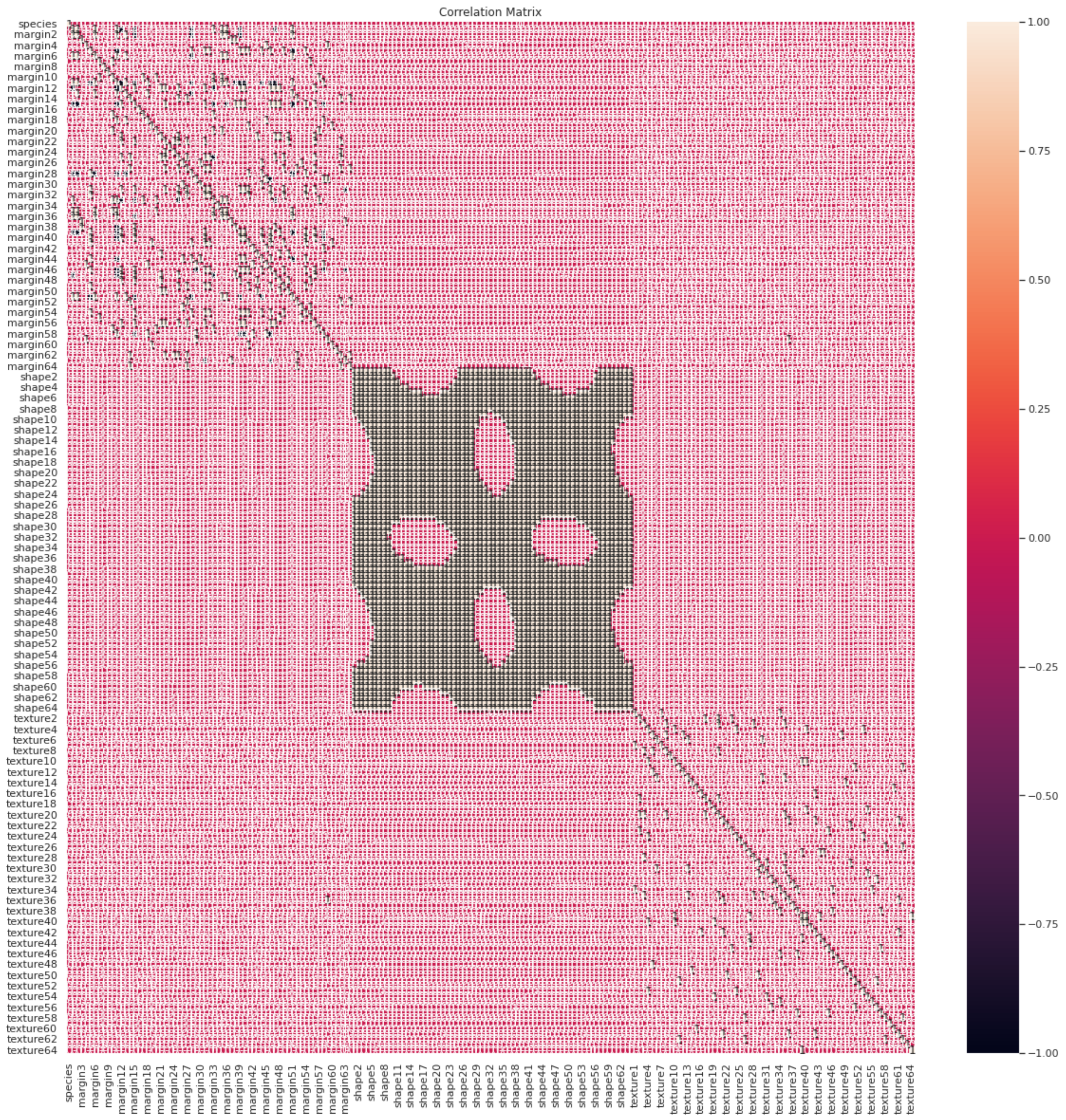
We made heatmaps for correlations between:

1. All features together
2. Margin_columns
3. Shape_columns
4. Texture_columns
5. Margin_columns vs Shape_columns

6. Margin_columns vs Texture_columns
7. Shape_columns vs Texture_columns

And found out that there is no high correlation between features except for shape columns which have many pairs with correlation up to 0.99.
And that is a sign that we can use feature reduction on shape columns (ex. PCA)

In the correlation matrix there seemed to be some kind of pattern between the shapes, this could be due to all the shapes having similar features (all leafs share similar center and common background



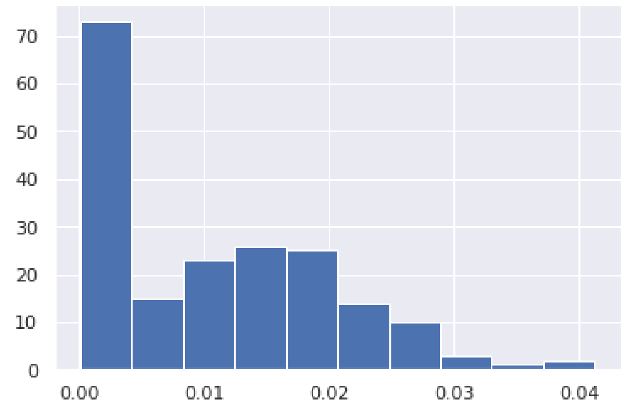
5. Split the data

Data was splitted using sklearn train_test_split with shuffle the data and split the using test size of 0.2
Then converted the target column (species) to categorical.

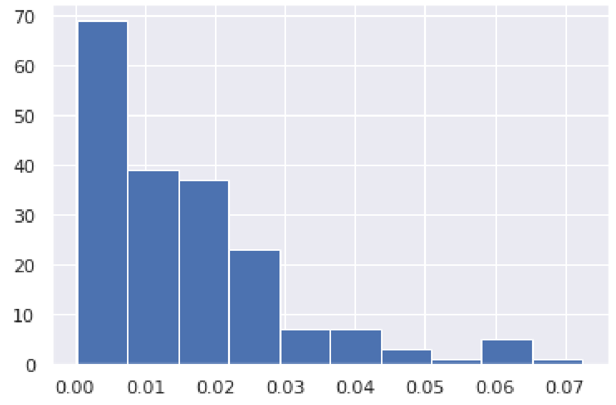
6. Decide if you need to standardize

Made 2 plots the first one is the histogram of means of features:

```
# x-axis: bins containing mean value
# y-axis: number of features with that mean
```



```
# x-axis: bins containing std value
# y-axis: number of features with that std
```



From the previous figure we can conclude that not all features have mean of zero
And no feature has std on 1 (std ranges from 0.01 to 0.07) so we need to standardize as that will help in our model training and make the model converge faster.

7. Modeling

7.1. First trial: Hidden size and Dropout

For the first trial we tried 3 hidden size (number of neurons in hidden layer), but after fixing the hidden size and tuning the dropout rate we realized that for each number of hidden units there is a corresponding value for the optimal dropout rate
So, we had to search for the best number of neurons in hidden layer with the dropout rate simultaneously

The result of this trial represented in the following table

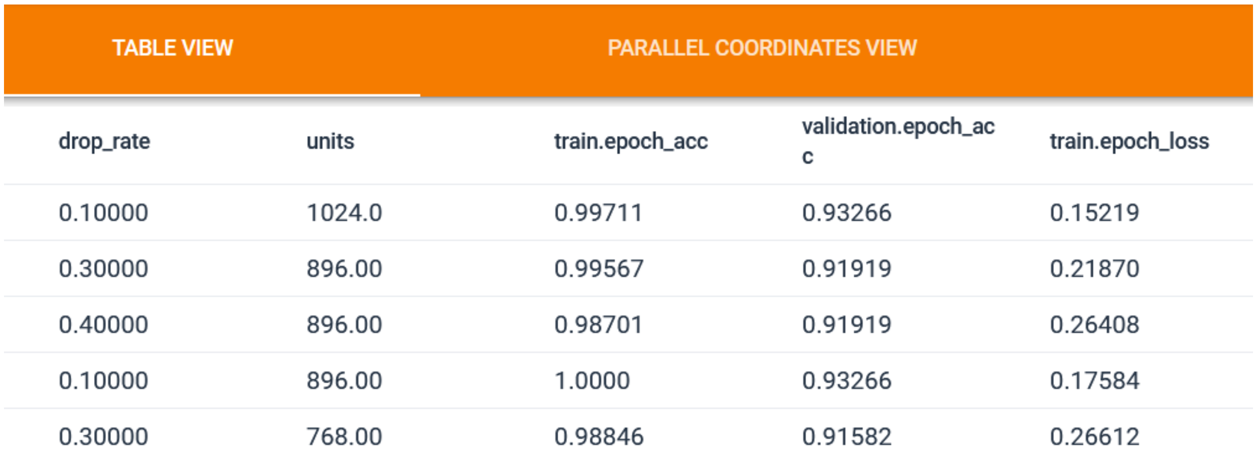
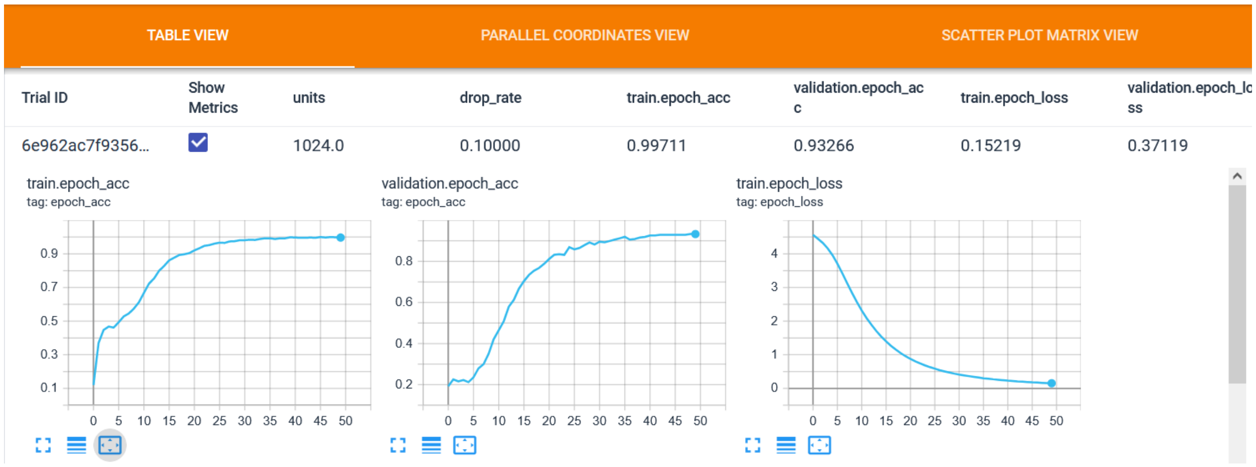


TABLE VIEW
PARALLEL COORDINATES VIEW

drop_rate	units	train.epoch_acc	validation.epoch_acc	train.epoch_loss
0.10000	1024.0	0.99711	0.93266	0.15219
0.30000	896.00	0.99567	0.91919	0.21870
0.40000	896.00	0.98701	0.91919	0.26408
0.10000	896.00	1.0000	0.93266	0.17584
0.30000	768.00	0.98846	0.91582	0.26612

As the table shows, there is a tie between units 1024 and 896 with the same value of the dropout rate which equals 0.1.
Despite that 1024 has lower training accuracy than 896, using 1024 units will reduce the generalization gap ,and also minimize the loss.

tensorboard of trial 1 :
https://tensorboard.dev/experiment/01GcGpkVTECMLjsOOXL7cg/#hparams



Trial ID	Show Metrics	units	drop_rate	train.epoch_acc	validation.epoch_acc	train.epoch_loss	validation.epoch_loss
6e962ac7f9356...	✓	1024.0	0.10000	0.99711	0.93266	0.15219	0.37119

7.2. Second trial: The best optimizer

For the second trial we use the fixed value for the best hidden size and dropout rate which we got from the previous trial and searching for the best optimizer

The result of this trial represented in the comparison between different optimizer as shown in the following table

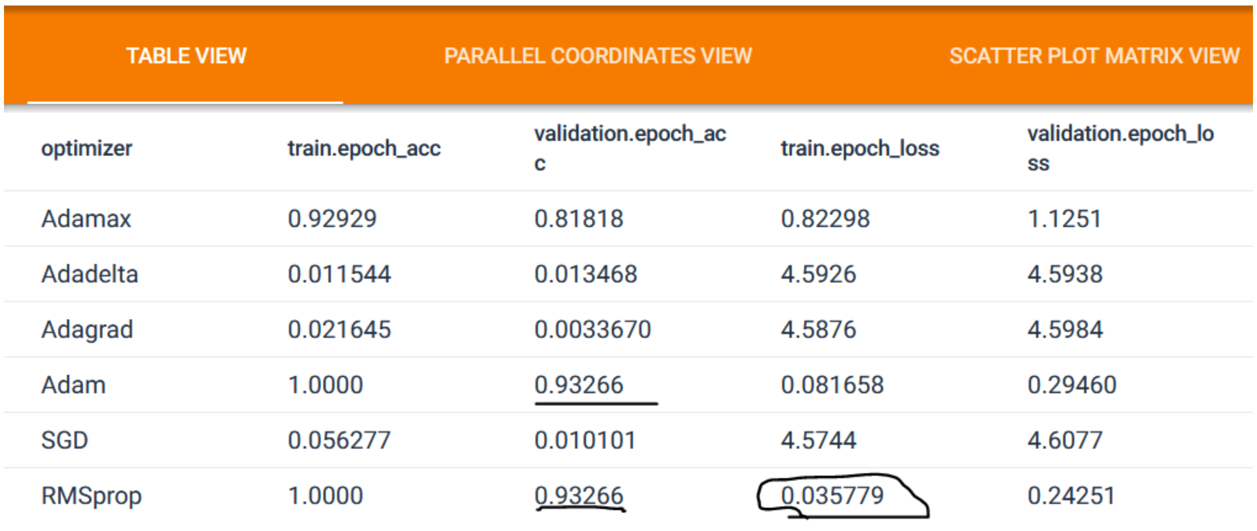


TABLE VIEW
PARALLEL COORDINATES VIEW
SCATTER PLOT MATRIX VIEW

optimizer	train.epoch_acc	validation.epoch_acc	train.epoch_loss	validation.epoch_loss
Adamax	0.92929	0.81818	0.82298	1.1251
Adadelta	0.011544	0.013468	4.5926	4.5938
Adagrad	0.021645	0.0033670	4.5876	4.5984
Adam	1.0000	0.93266	0.081658	0.29460
SGD	0.056277	0.010101	4.5744	4.6077
RMSprop	1.0000	0.93266	0.035779	0.24251

tensorboard of comparing all the optimizer:
https://tensorboard.dev/experiment/gfnaduZ5RDePEMzZrWi9qw/#hparams

In this trial there is a tie also between Adam and RMSprop, but RMSprop has lower loss.

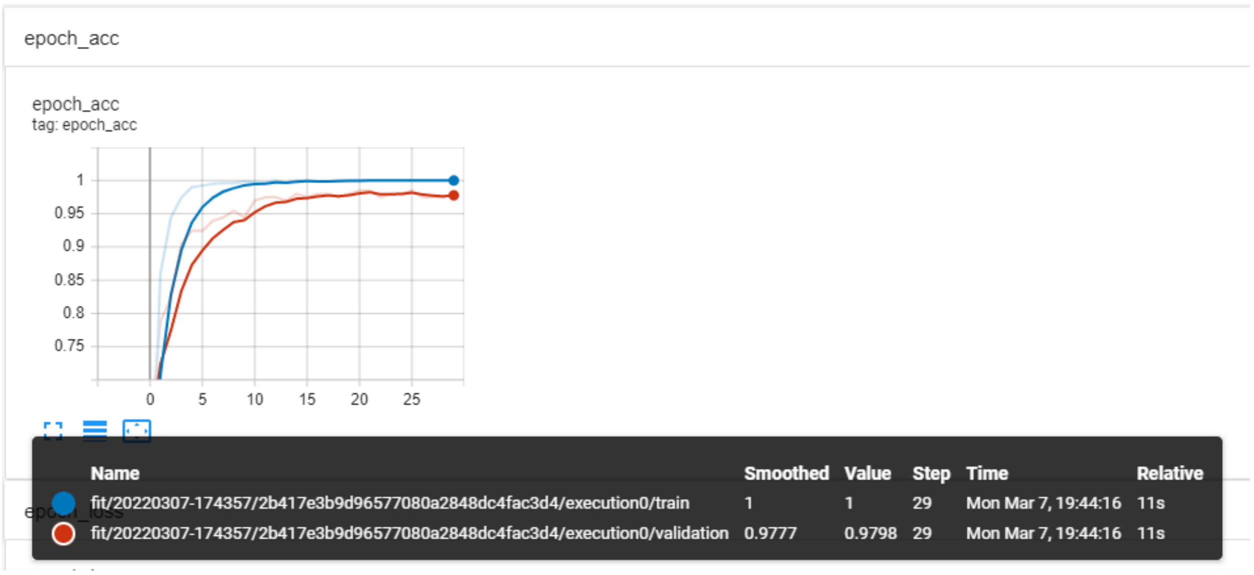
7.3. Third trial

For the third trial we use the fixed value for the best hidden size, the dropout rate, and the optimizer which we got from the previous trials and trying to add and remove the batch norm

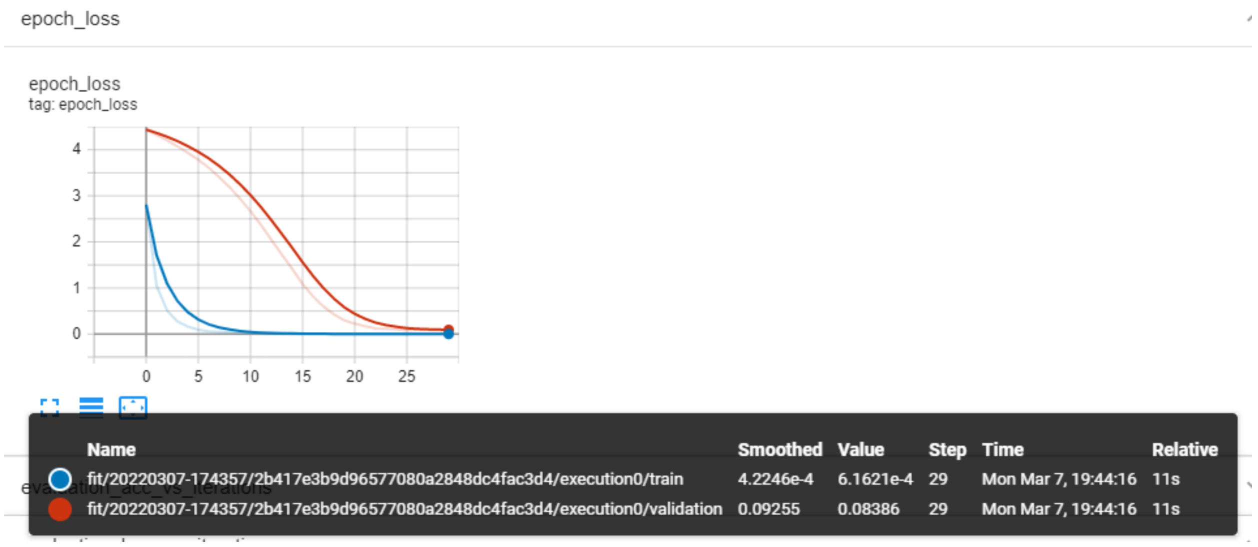
The distribution of the inputs to layers deep in the network may change after each mini-batch step, This can cause the model to forever chase a moving target and take longer time, this could be more noticeable in deeper neural network

Even though in our case we only have one hidden layer it helped in converging faster and reaching a higher validation accuracy

tensorboard of adding batch norm:
https://tensorboard.dev/experiment/f4ggykCPRa28MvJWqjgDRQ/

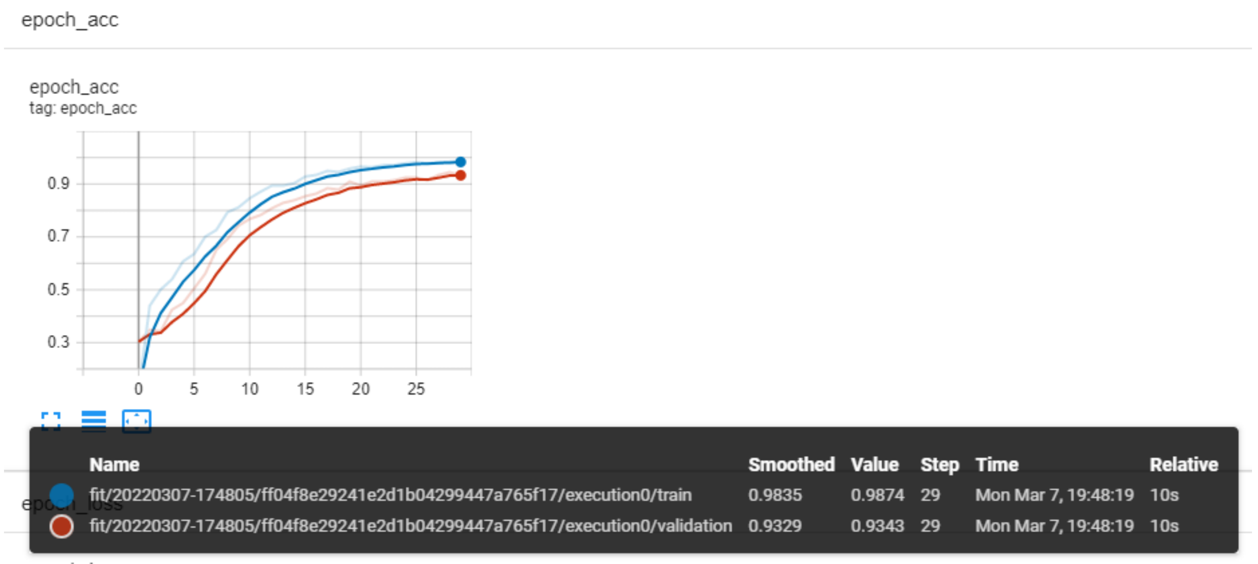


	Name	Smoothed	Value	Step	Time	Relative
●	fit/20220307-174357/2b417e3b9d96577080a2848dc4fac3d4/execution0/train	1	1	29	Mon Mar 7, 19:44:16	11s
●	fit/20220307-174357/2b417e3b9d96577080a2848dc4fac3d4/execution0/validation	0.9777	0.9798	29	Mon Mar 7, 19:44:16	11s

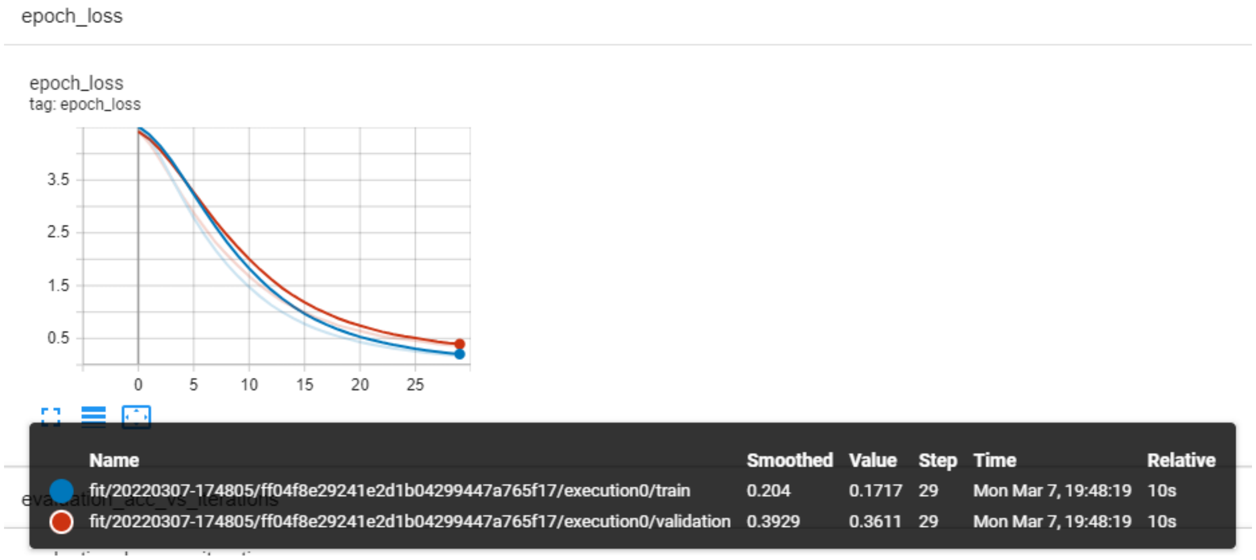


	Name	Smoothed	Value	Step	Time	Relative
🔵	fit/20220307-174357/2b417e3b9d96577080a2848dc4fac3d4/execution0/train	4.2246e-4	6.1621e-4	29	Mon Mar 7, 19:44:16	11s
🔴	fit/20220307-174357/2b417e3b9d96577080a2848dc4fac3d4/execution0/validation	0.09255	0.08386	29	Mon Mar 7, 19:44:16	11s

tensorboard of model without batch norm:
https://tensorboard.dev/experiment/Zpr1zXfzQVankgpyTYDHyQ/



	Name	Smoothed	Value	Step	Time	Relative
🔵	fit/20220307-174805/ff04f8e29241e2d1b04299447a765f17/execution0/train	0.9835	0.9874	29	Mon Mar 7, 19:48:19	10s
🔴	fit/20220307-174805/ff04f8e29241e2d1b04299447a765f17/execution0/validation	0.9329	0.9343	29	Mon Mar 7, 19:48:19	10s



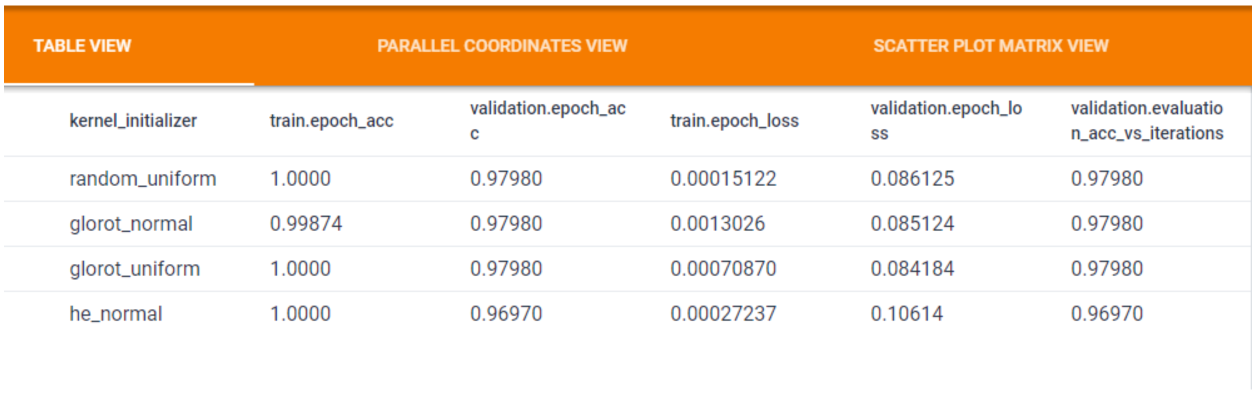
	Name	Smoothed	Value	Step	Time	Relative
●	fit/20220307-174805/ff04f8e29241e2d1b04299447a765f17/execution0/train	0.204	0.1717	29	Mon Mar 7, 19:48:19	10s
●	fit/20220307-174805/ff04f8e29241e2d1b04299447a765f17/execution0/validation	0.3929	0.3611	29	Mon Mar 7, 19:48:19	10s

running the same model without batchnorm resulted a lower accuracy and the model converged slower for the same step (91% validation accuracy in 80 steps , compared to 96% in the same step)

7.4. Fourth trial: Kernel initializer

For the fourth trial we try to find the best kernel initializer and best learning rate using the fixed value for the best hyperparameters that we get from all the previous trials

tensorboard of model with different kernel initializers : https://tensorboard.dev/experiment/EKN2FVWWTvKARgA8XOqV4A/#hparams



kernel_initializer	train.epoch_acc	validation.epoch_acc	train.epoch_loss	validation.epoch_loss	validation.evaluation_acc_vs_iterations
random_uniform	1.0000	0.97980	0.00015122	0.086125	0.97980
glorot_normal	0.99874	0.97980	0.0013026	0.085124	0.97980
glorot_uniform	1.0000	0.97980	0.00070870	0.084184	0.97980
he_normal	1.0000	0.96970	0.00027237	0.10614	0.96970

In this trial there is a tie also between random_uniform, glorot_normal and glorot_uniform, but glorot_normal was less overfitting

7.5. Fifth trial: standard scaler and regularization

In this trial we will apply StandardScaler on input data which will compute the mean and standard deviation for each feature dimension using the training set only, then subtracting the mean and dividing by the stdev for each feature and each sample.
tensorboard of model:
https://tensorboard.dev/experiment/4uPlrrL3SXOBqjawGhDkXQ/
And we achieved Validation acc = 98.6

8. Conclusion

Out of all the trials trial 1 had the most impact in training accuracy , and it was very sensitive to change, the possibilities can vary more than we can cover, and the results could be improved be running finer grained search (gridsearch) and larger scope of hidden neuron
Best Train Acc : 0.997 , test_acc: 0.932

For trial 2 both RMSprop and Adam had the same result with slight improvements when using RMSprop
Performance improved slightly Train Acc : 1.0 , test_acc: 0.93266 (overfitting)

For trial 3 batch normalization had noticeable impact in the convergence speed and resulted a better accuracy
Performance improved slightly Train Acc : 1.0 , test_acc: 0.9663

For trial 4 kernel initializer didn't have much contribution in model accuracy but glorot_normal was less overfitting.
Performance improved slightly Train Acc : 0.99874 , test_acc: 0.97980

For trial 5 standard scaler and regularization have great effect on scores, the scores improved and achieved a score of val_acc **98.6**

The data is highly cleaned and went through huge amount of preprocessing which made the training fairly simple for a neural network, the most unexpected outcome was how much kernel initializers could make a difference