The Project Report

Team Members:

Ahmed Salama (6)

Zeyad Tarek (55)

Mahmoud Khorshed (29)

Supervised by:

Prof. Hazem Abbas

Project objective:

- In this project, you will use the Leaf Classification dataset using a neural network architecture:

Problem Formulation

- **Input**: - Features collected from half a million species of plant in the world.0
- **Output**: - Predicted species for leaves.
- **Deep Learning Function**: - Manipulating, analyzing, preprocessing the data, and training the data.
- **Problems**: - Classification of species has been historically problematic and often results in duplicate identifications.
- **Objective**: - The objective of this playground competition is to use binary leaf images and extracted features, including shape, margin & texture, to accurately identify 99 species of plants. Leaves, due to their volume, prevalence, and unique characteristics, are an effective means of differentiating plant species.
- **Challenges ▶** :
  1. Nan cells.
  2. Unused and unimportant column.
  3. Convert the type of some features.
  4. convert strings by One Hot encoding.
  5. Choose the best hyper parameters for the network.
- **Impact ▶**: Predicting the species of the leaf that will lead to a successful match.

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 and finally, it contains **193 Features**.
- Note that of the original 100 species, we have eliminated one on account of incomplete associated data in the original dataset.

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 Exploratory:

1- Describe the data using summary statistics.

- The data is **normalized, so we won’t use standard scaler.**

The describe() function in pandas is very handy in getting various summary statistics.This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

```
[ ]  all_data.describe()
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	te
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	...	1584.000000	1584.000000	158
mean	0.017468	0.028497	0.031939	0.023008	0.014362	0.038174	0.019209	0.001084	0.007139	0.018699	...	0.036047	0.005361	
std	0.019675	0.038655	0.025791	0.028550	0.018250	0.051771	0.017361	0.002725	0.009153	0.016126	...	0.063792	0.022476	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	0.005859	...	0.000000	0.000000	
50%	0.009766	0.011719	0.023438	0.013672	0.007812	0.013672	0.015625	0.000000	0.005859	0.015625	...	0.003906	0.000000	
75%	0.027344	0.041016	0.044922	0.029297	0.019531	0.056641	0.029297	0.000000	0.007812	0.027344	...	0.041260	0.000000	
max	0.087891	0.205080	0.167970	0.169920	0.111330	0.310550	0.091797	0.031250	0.083984	0.097656	...	0.429690	0.441410	

8 rows × 192 columns

2- Correlation Analysis: -

- Sample of the correlation.

```
corr = all_data.corr()

corr.style.background_gradient(cmap="Spectral")
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	margin12	margin13	margin14	margin15	margin16	margin17	ma
margin1	1.000000	0.806390	-0.182829	-0.297807	-0.475874	0.767718	0.066273	-0.094137	-0.181496	0.397138	0.737461	-0.528224	0.489317	-0.370460	-0.540974	-0.072127	0.316704	0.
margin2	0.806390	1.000000	-0.204640	-0.315953	-0.444312	0.825762	-0.083273	-0.086428	-0.120276	0.162587	0.805064	-0.489808	0.647166	-0.316377	-0.503059	-0.068356	0.135000	0.
margin3	-0.182829	-0.204640	1.000000	0.120042	-0.185007	-0.163976	0.095449	0.024350	-0.000042	0.008772	-0.261371	-0.004085	-0.048698	0.095701	0.050113	-0.054076	-0.130220	-0.
margin4	-0.297807	-0.315953	0.120042	1.000000	0.029480	-0.261437	-0.268271	-0.047693	0.227543	-0.173986	-0.172503	-0.202576	-0.238041	0.338136	-0.259813	-0.021615	-0.047704	0.
margin5	-0.475874	-0.444312	-0.185007	0.029480	1.000000	-0.438587	-0.108178	0.056557	0.196745	-0.320647	-0.514981	0.373683	-0.463328	0.095697	0.467991	0.081766	-0.235063	-0.
margin6	0.767718	0.825762	-0.163976	-0.261437	-0.438587	1.000000	-0.093780	-0.112896	-0.136961	0.215141	0.686998	-0.479464	0.539807	-0.317465	-0.489144	-0.065768	0.120157	0.
margin7	0.066273	-0.083273	0.095449	-0.268271	-0.108178	-0.093780	1.000000	0.099867	-0.350804	0.649311	-0.069978	-0.144810	-0.116093	-0.357485	0.004146	-0.023989	0.396388	-0.
margin8	-0.094137	-0.086428	0.024350	-0.047693	0.056557	-0.112896	0.099867	1.000000	-0.071887	0.012918	-0.108453	0.044335	-0.049359	0.001100	0.062293	0.205817	0.025698	-0.
margin9	-0.181496	-0.120276	-0.000042	0.227543	0.196745	-0.136961	-0.350804	-0.071887	1.000000	-0.337466	-0.139592	-0.065846	-0.053170	0.372013	-0.117375	-0.026071	-0.236991	-0.
margin10	0.397138	0.162587	0.008772	-0.173986	-0.320647	0.215141	0.649311	0.012918	-0.337466	1.000000	0.226220	-0.384797	0.043592	-0.416157	-0.320361	-0.053492	0.566792	0.
margin11	0.737461	0.805064	-0.261371	-0.172503	-0.514981	0.686998	-0.069978	-0.108453	-0.139592	0.226220	1.000000	-0.579610	0.665286	-0.373834	-0.603805	-0.081410	0.226382	0.
margin12	-0.528224	-0.489808	-0.004085	-0.202576	0.373683	-0.479464	-0.144810	0.044335	-0.065846	-0.384797	-0.579610	1.000000	-0.429504	0.207537	0.806811	0.061776	-0.340929	-0.
margin13	0.489317	0.647166	-0.048698	-0.238041	-0.463328	0.539807	-0.116093	-0.049359	-0.053170	0.043592	0.665286	-0.429504	1.000000	-0.274913	-0.441614	-0.068353	0.006105	0.

3- Image discovery (show some images and their types)

Acer_Opalus

Pterocarya_Stenoptera

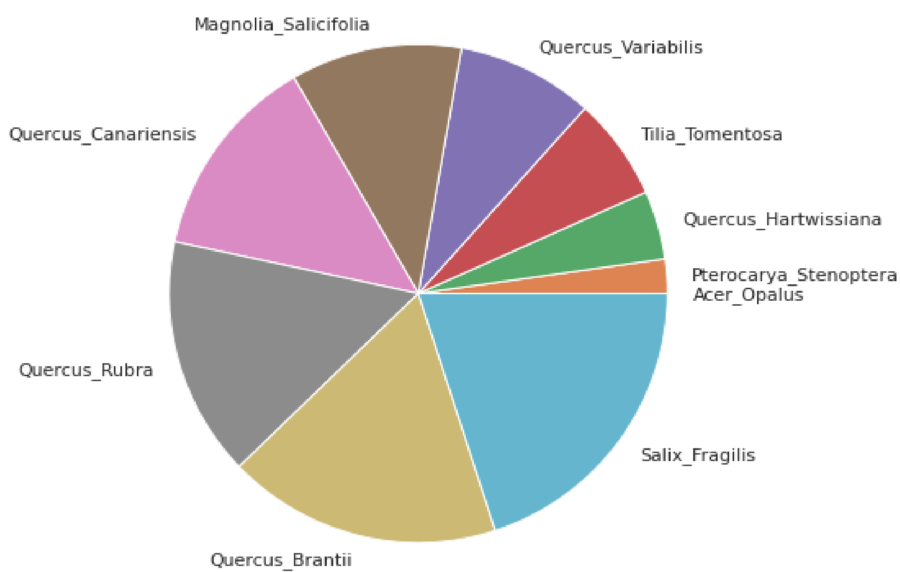
Quercus_Hartwissiana

Tilia_Tomentosa

Part 2: Data Preprocessing:

- Check duplication: - We found no duplicates.
- Check missing values: We found no null values.
- We didn’t scale our data because the range of the data is too small.
- Correlation analysis: We drop correlated features with more than absolute value = .95, we discovered that there are more 63 features were removed (almost from shapes).
- We have split the data into 80% training
- Encode the labels using factorize

- Some visualizations:



Part III: Build a neural network and tuning its hyperparameters**:**

We used Keras tuner to choose the best hyperparameters.

Tuned hyperparameters

- **`hidden units`**
- **`hidden dropout`**
- **`l1_penalty_hidden`**
- **`l2_penalty_hidden`**
- **`l2_penalty_hidden_bias`**
- **`Early Stopping`**
- **`Batch size`**

Step 1 - the search method used to find best combination of hyperparameters is **hyperband** (it's faster than **bayesian search** and better than **random search**).

Step 2 - After running so many trials searching for best hyperparameters we selected the best 5 hyperparameters combination that get best validation accuracy score.

Here's the five models and their hyperparameters:

	Learning rate	Best hidden units number	Best hidden units L2	Best hidden L1	Best hidden L2 bias	Best Dropout Rate
0	0.017807	416	0.0000	0.0000	0.0030	0.25
1	0.055227	256	0.0000	0.0015	0.0000	0.05
2	0.006341	320	0.0015	0.0000	0.0030	0.00
3	0.054649	224	0.0000	0.0015	0.0045	0.20
4	0.003193	448	0.0015	0.0000	0.0000	0.25

Step 3- After that we took 5 hyperparameters combination and build new 5 models for one of them, and each model of them has input layer and hidden layer with **activation function tanh** and output layer

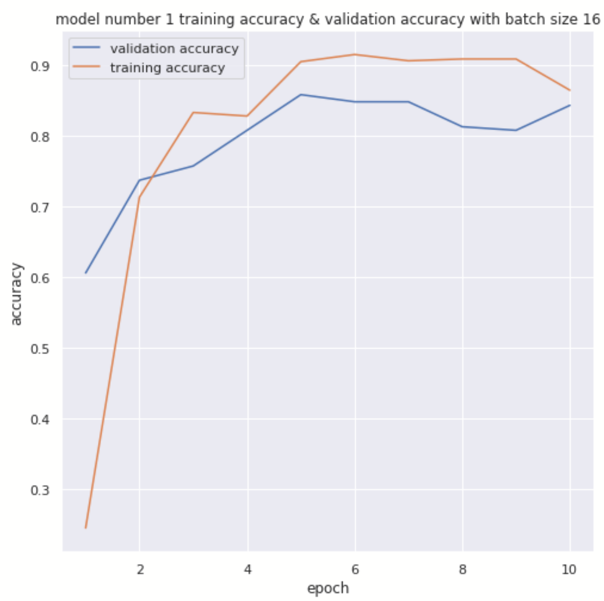
Part IV: Train our neural networks**:**

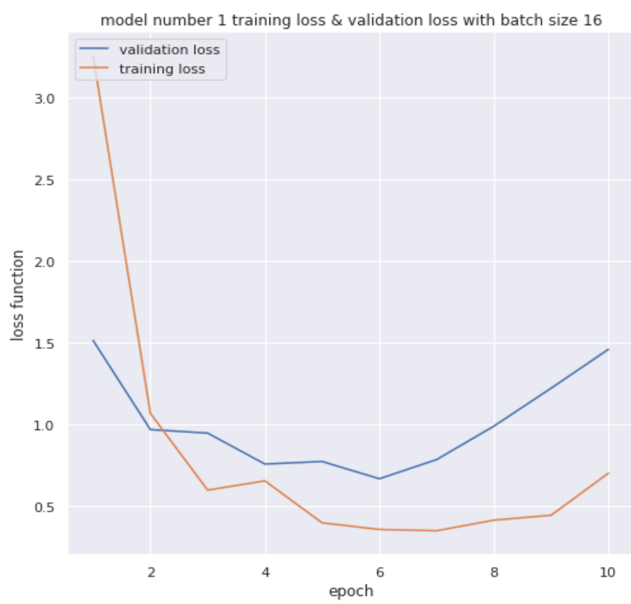
Step 1: By making a list that have all unfitted models.

Step 2: Start training each model using **‘for loop’** to over loop the models list and take care consider the early stopping point.

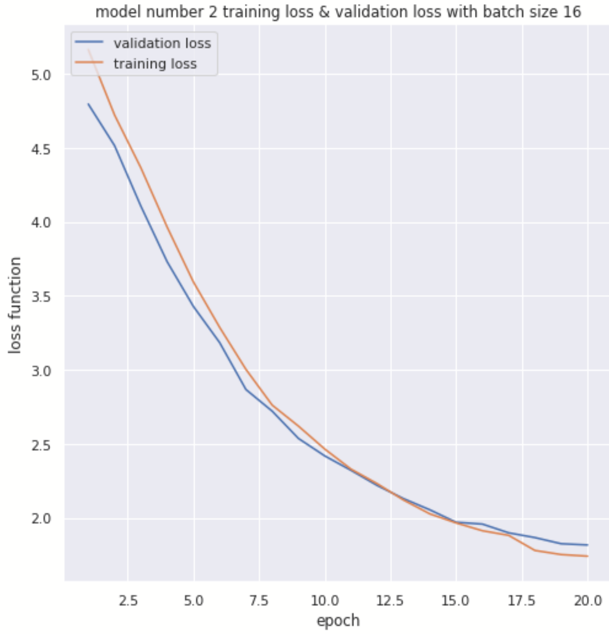
Step 3- Plot each model **training accuracy/loss** and **validation accuracy/loss** for each model and plot them over the number of epochs.

Model number 1 with batch size 16:

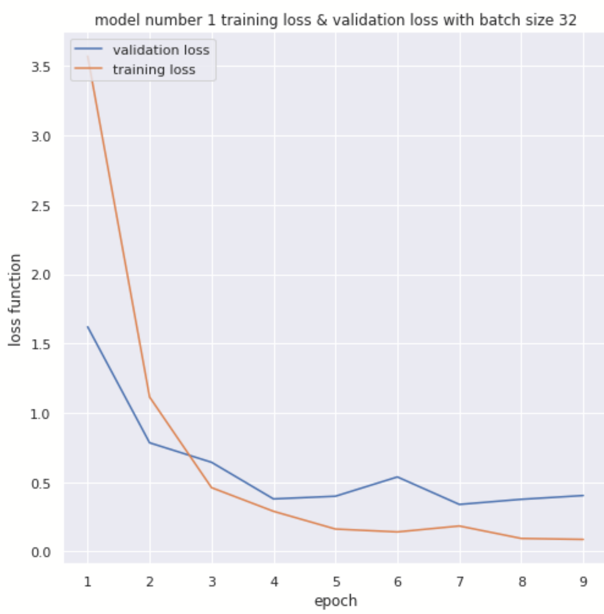




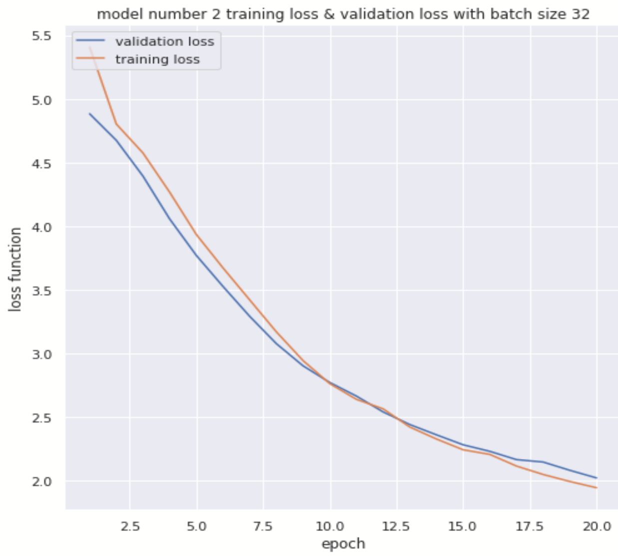
Model number 2 with batch size 16

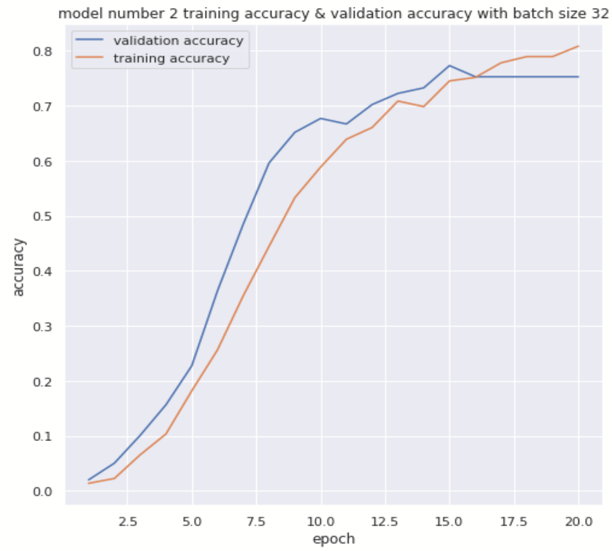


Model number 1 with batch size 32:

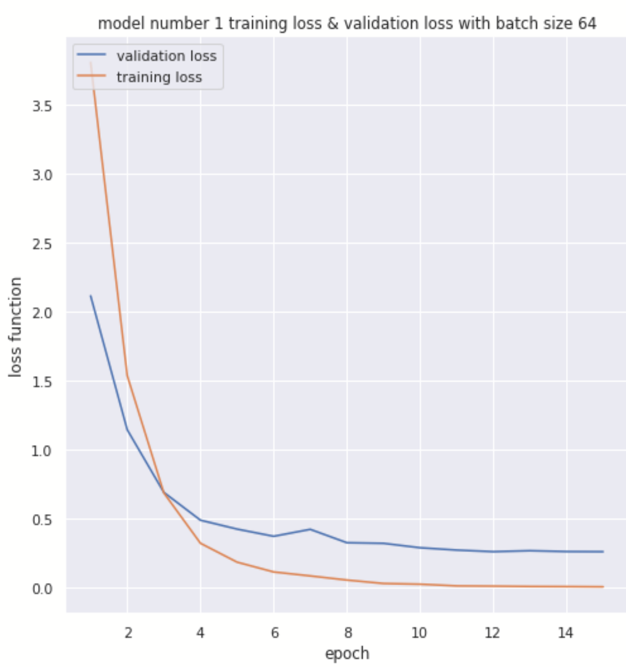


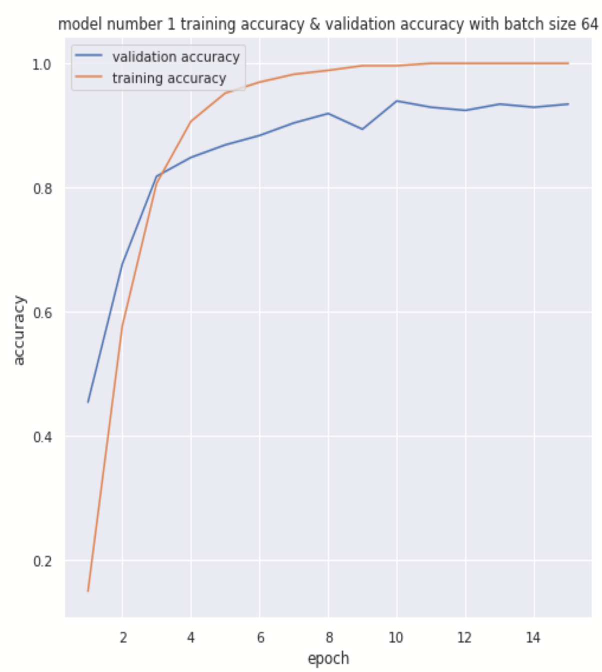
Model number 2 with batch size 32:



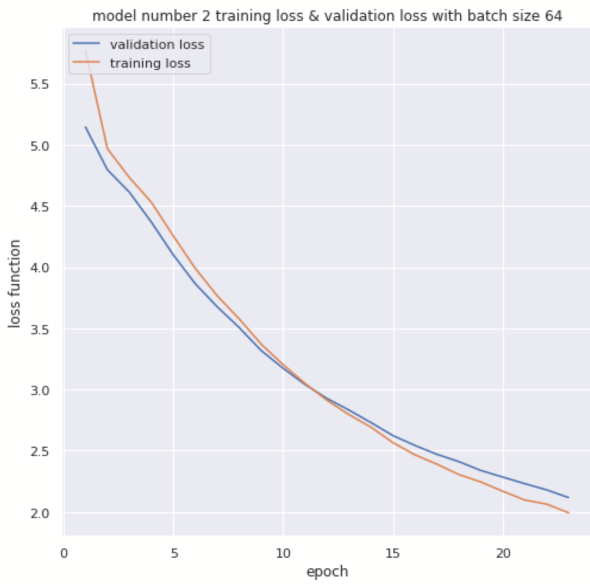


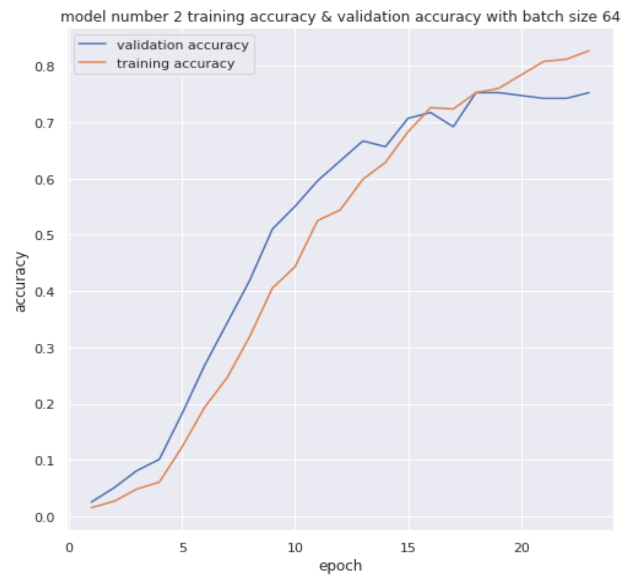
Model number 1 with batch size 64:





Model number 2 with batch size 64

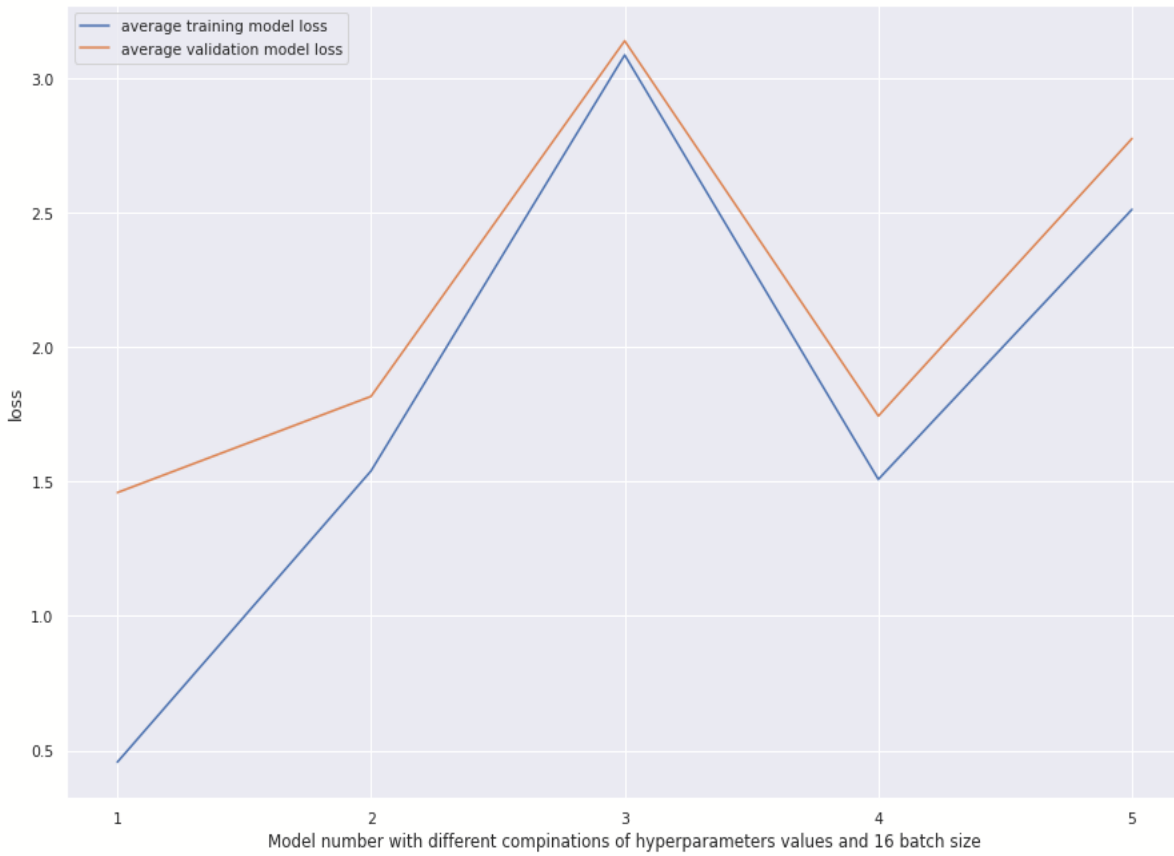




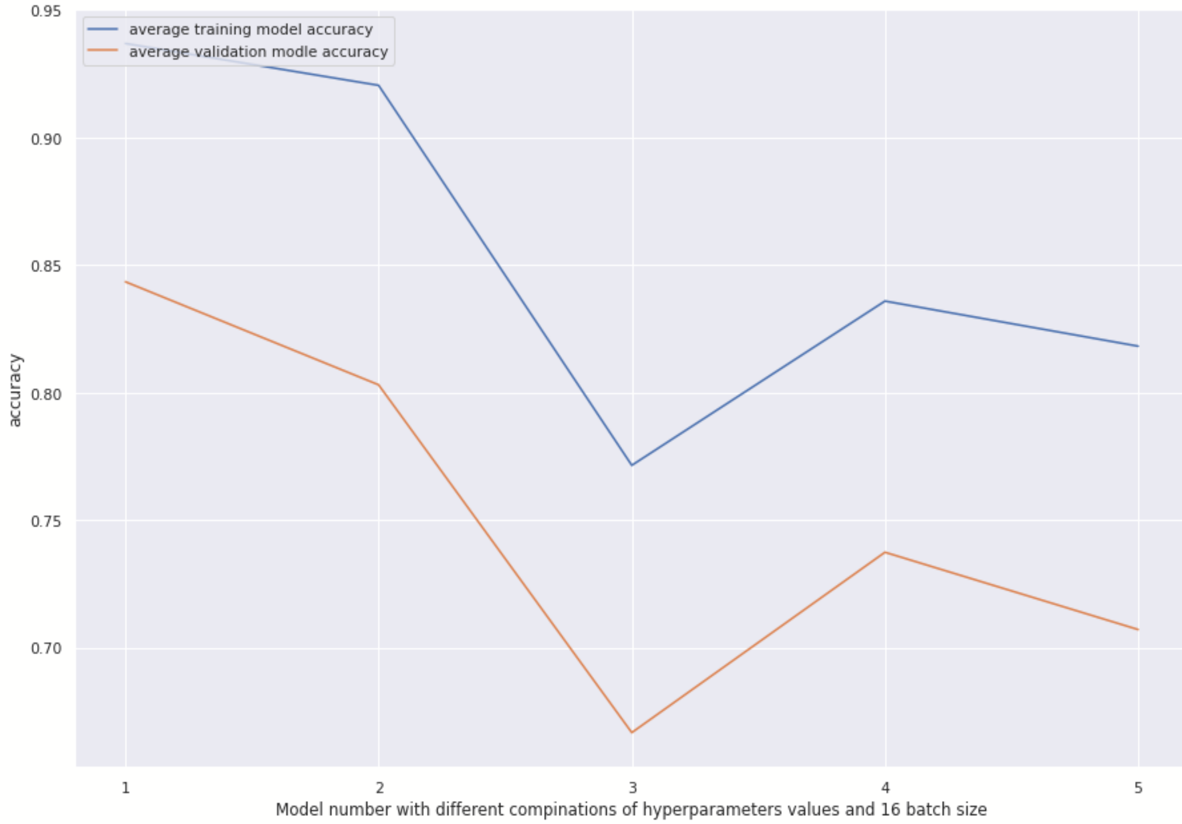
There are still 3 models plots for each batch size you can go and see them in the code.

5- We will evaluate each model and we save each trained model's average training and validation accuracy/loss to plot them.

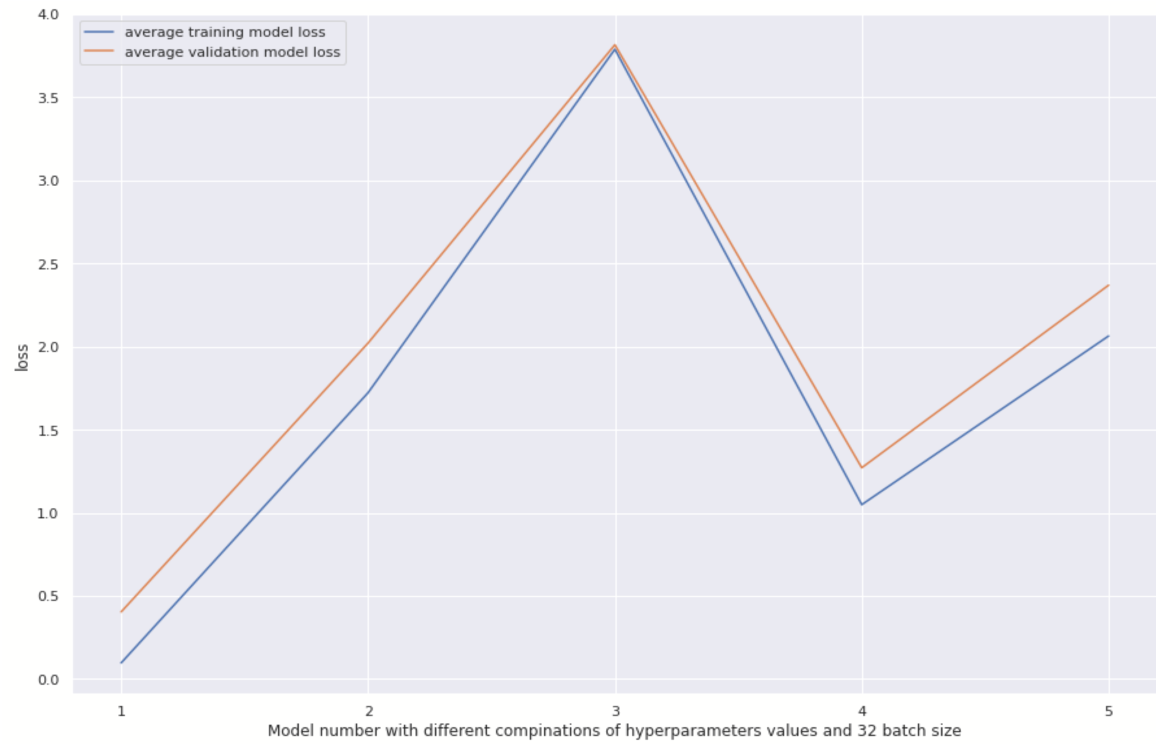
- ***16 Batch Size – Average Training Loss Vs Average Validation loss for each model***



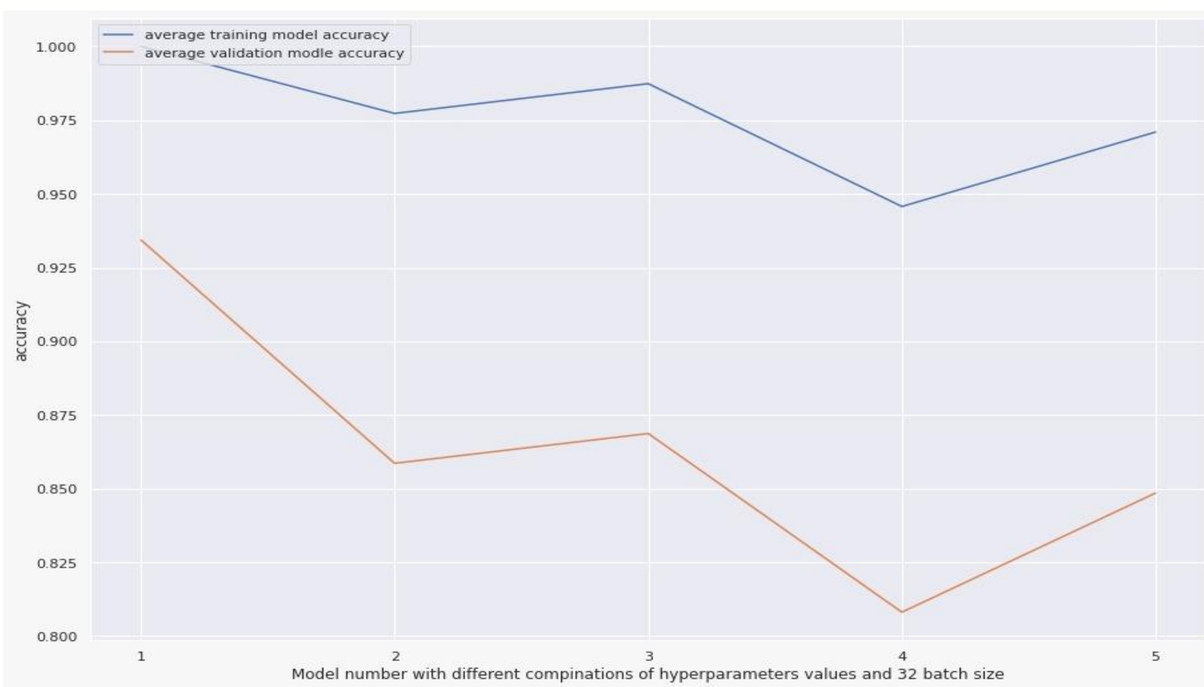
- ***16 Batch Size – Average Training Accuracy Vs Average Validation Accuracy for each model***



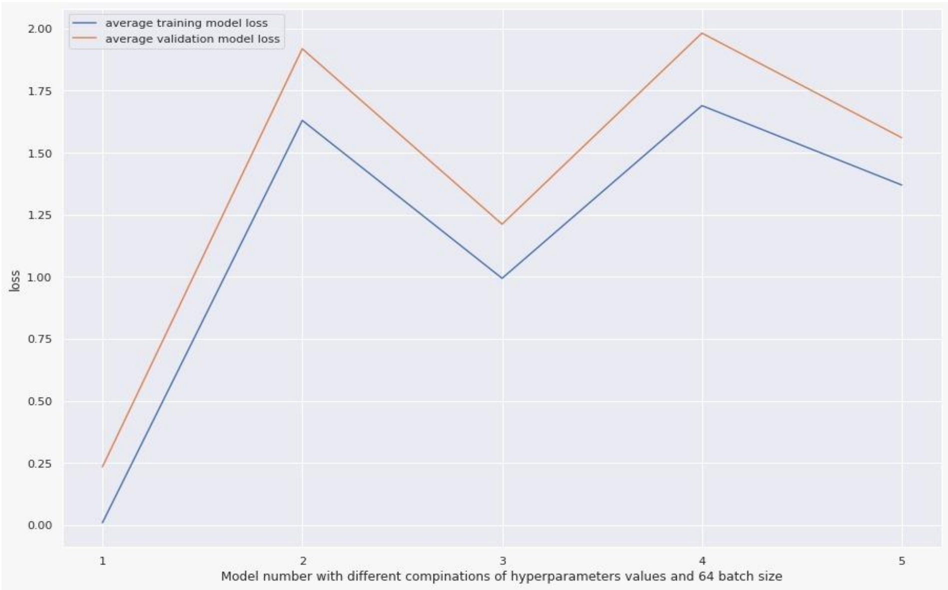
- ***32 Batch Size – Average Training Loss Vs Average Validation loss***



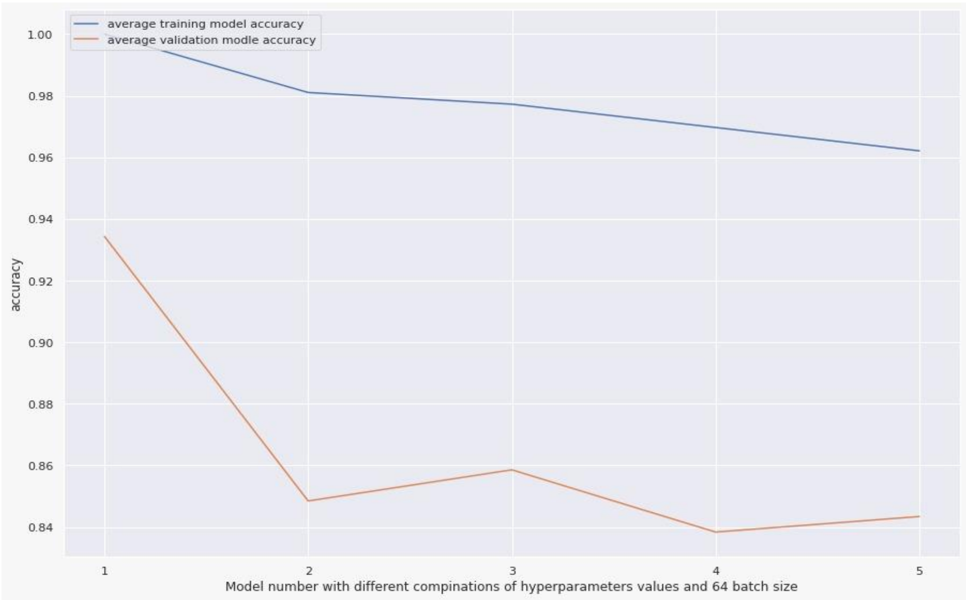
- ***32 Batch Size – Average Training Accuracy Vs Average Validation Accuracy***



- ***64 Batch Size – Average Training loss Vs Average Validation loss***



- ***64 Batch Size – Average Training Accuracy Vs Average Validation Accuracy***



Final results:

After training 15 models and test them on unseen data '0.2' we got these results:

	Loss	accuracy
Model 1 with batch size 16	1.459202766418457	84.34%
Model 2 with batch size 16	1.817238450050354	80.30%
Model 3 with batch size 16	3.1403920650482178	66.66%
Model 4 with batch size 16	1.7442669868469238	73.72%
Model 5 with batch size 16	2.7768640518188477	70.71%
Model 1 with batch size 32	0.40393945574760437	90.40%
Model 2 with batch size 32	2.020517587661743	75.25%
Model 3 with batch size 32	3.8146913051605225	65.15%
Model 4 with batch size 32	1.2708925008773804	85.35%
Model 5 with batch size 32	2.3690359592437744	76.76%
Model 1 with batch size 64	**0.2606773376464844**	**93.43%**
Model 2 with batch size 64	2.1163594722747803	75.25%
Model 3 with batch size 64	3.9199368953704834	65.65%
Model 4 with batch size 64	1.3419238328933716	84.84%
Model 5 with batch size 64	2.423539638519287	75.25%

Conclusions:

from about results the winner model after creating and training 15 models is the model number 1 and that's natural because the first hyperparametersare the best and the ones who got best validation accuracy.

Best Hyperparameters:

1. Learning rate: 0.046368
2. Best hidden units number: 192
3. Best hidden units L2: 0.0000
4. Best hidden L1: 0.0000
5. Best hidden L2 bias: 0.0030
6. Best Dropout Rate: 0.0015
7. Best batch size: 64

Best validation accuracy is 93.43%

Could we get best validation accuracy?

Yes, by increasing the number of max_epochs in 'kt.tuner()' and the number of epochs in tuner.search but this will increase the time of searching.

References

- *https://www.tensorflow.org/tutorials/keras/keras_tuner*