CISC 867 - Deep Learning

Project 1: Leaf classification.

Supervised by:

Dr. Hazem Abbas

T.A. Asef Mahfouz

T.A. Reem Abdel-salam

Name: Manar Samy Elghobashy

Id: 20398569

problem definition:

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.

This problem aims to identify 99 species of plants due to their volume, prevalence, and unique characteristics, which are an effective means of differentiating plant species.

problem formulation:

1. input: Features collected from half a million species of plant in the world including shape, margin & texture.

2. output: predicted species.

3. deep learning: fully connected neural networks with different hyperparameters.

Data description:

Train.csv contains 990 records with 194 features including id, species, margin 1, etc.

Data features:

- 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

Reading train/test data:

```
#reading train data csv file.
train_data=pd.read_csv('train.csv/train.csv')
train_data.head()
```

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58	textu
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	...	0.007812	0.000000	0.002930	0.002930	0.035
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	...	0.000977	0.000000	0.000000	0.000977	0.023
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	...	0.154300	0.000000	0.005859	0.000977	0.007
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	...	0.000000	0.000977	0.000000	0.000000	0.020
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	...	0.096680	0.000000	0.021484	0.000000	0.000

5 rows × 194 columns

```
#reading test data csv file.
test_data=pd.read_csv('test.csv/test.csv')
test_data.head()
```

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56	texture57	texture58	texture59	texture
0	4	0.019531	0.009766	0.078125	0.011719	0.003906	0.015625	0.005859	0.0	0.005859	...	0.006836	0.000000	0.015625	0.000977	0.015625	
1	7	0.007812	0.005859	0.064453	0.009766	0.003906	0.013672	0.007812	0.0	0.033203	...	0.000000	0.000000	0.006836	0.001953	0.013672	
2	9	0.000000	0.000000	0.001953	0.021484	0.041016	0.000000	0.023438	0.0	0.011719	...	0.128910	0.000000	0.000977	0.000000	0.000000	
3	12	0.000000	0.000000	0.009766	0.011719	0.017578	0.000000	0.003906	0.0	0.003906	...	0.012695	0.015625	0.002930	0.036133	0.013672	
4	13	0.001953	0.000000	0.015625	0.009766	0.039062	0.000000	0.009766	0.0	0.005859	...	0.000000	0.042969	0.016602	0.010742	0.041016	

5 rows × 193 columns

Part I: Data Preparation.

1. Describe the data.
   By using describe () function a detailed description of the data like count, mean, std, etc.

```
# describtion of the data.
train_data.describe()
```

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56	
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	...	990.000000	990.000000	9
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	...	0.036501	0.005024	
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	...	0.063403	0.019321	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	...	0.000000	0.000000	
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	...	0.004883	0.000000	
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	...	0.043701	0.000000	
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	...	0.429690	0.202150	

8 rows × 193 columns

2. Check null and duplication in the data:
   Using isnull () and duplicated () functions to check if there are any null or missing and duplicated data.

```
#check if there is any null value.
train_data.isnull().sum()
```

```
id           0
species      0
margin1      0
margin2      0
margin3      0
            ..
texture60    0
texture61    0
texture62    0
texture63    0
texture64    0
Length: 194, dtype: int64
```

```
#check if there is any duplications
train_data.duplicated().sum()
```

```
0
```

3. Explore the species ‘label’ column:
   Here, a unique () function is used to display unique values in the label column species.

```
# see species unique values.
train_data['species'].unique()
```

```
array(['Acer_Opalus', 'Pterocarya_Stenoptera', 'Quercus_Hartwissiana',
       'Tilia_Tomentosa', 'Quercus_Variabilis', 'Magnolia_Salicifolia',
       'Quercus_Canariensis', 'Quercus_Rubra', 'Quercus_Brantii',
       'Salix_Fragilis', 'Zelkova_Serrata', 'Betula_Austrosinensis',
       'Quercus_Pontica', 'Quercus_Afares', 'Quercus_Coccifera',
       'Fagus_Sylvatica', 'Phildelphus', 'Acer_Palmatum',
       'Quercus_Pubescens', 'Populus_Adenopoda', 'Quercus_Trojana',
       'Alnus_Sieboldiana', 'Quercus_Ilex', 'Arundinaria_Simonii',
       'Acer_Platanoids', 'Quercus_Phillyraeoides', 'Cornus_Chinensis',
       'Liriodendron_Tulipifera', 'Cytisus_Battandieri',
       'Rhododendron_x_Russellianum', 'Alnus_Rubra',
       'Eucalyptus_Glaucescens', 'Cercis_Siliquastrum',
       'Cotinus_Coggygria', 'Celtis_Koraiensis', 'Quercus_Crassifolia',
       'Quercus_Kewensis', 'Cornus_Controversa', 'Quercus_Pyrenaica',
       'Callicarpa_Bodinieri', 'Quercus_Alnifolia', 'Acer_Saccharinum',
       'Prunus_X_Shmittii', 'Prunus_Avium', 'Quercus_Greggii',
       'Quercus_Suber', 'Quercus_Dolicholepis', 'Ilex_Cornuta',
       'Tilia_Oliveri', 'Quercus_Semecarpifolia', 'Quercus_Texana',
       'Ginkgo_Biloba', 'Liquidambar_Styraciflua', 'Quercus_Phellos',
       'Quercus_Palustris', 'Alnus_Maximowiczii', 'Quercus_Agrifolia',
       'Acer_Pictum', 'Acer_Rufinerve', 'Lithocarpus_Cleistocarpus',
       'Viburnum_x_Rhytidophylloides', 'Ilex_Aquifolium',
       'Acer_Circinatum', 'Quercus_Coccinea', 'Quercus_Cerris',
       'Quercus_Chrysolepis', 'Eucalyptus_Neglecta', 'Tilia_Platyphyllos',
       'Alnus_Cordata', 'Populus_Nigra', 'Acer_Capillipes',
       'Magnolia_Heptapeta', 'Acer_Mono', 'Cornus_Macrophylla',
       'Crataegus_Monogyna', 'Quercus_x_Turneri', 'Quercus_Castaneifolia',
       'Lithocarpus_Edulis', 'Populus_Grandidentata', 'Acer_Rubrum',
       'Quercus_Imbricaria', 'Eucalyptus_Urnigera', 'Quercus_Crassipes',
       'Viburnum_Tinus', 'Morus_Nigra', 'Quercus_Vulcanica',
       'Alnus_Viridis', 'Betula_Pendula', 'Olea_Europaea',
       'Quercus_Ellipsoidalis', 'Quercus_x_Hispanica',
       'Quercus_Shumardii', 'Quercus_Rhysophylla', 'Castanea_Sativa',
       'Ulmus_Bergmanniana', 'Quercus_Nigra', 'Salix_Intergra',
       'Quercus_Infectoria_sub', 'Sorbus_Aria'], dtype=object)
```

Display the number of unique values in the labels using the nuniques () function to know how many units will be in the output layer of the model. Then using value_counts, it displays the number of occurrences in each label.

```
#see the number of unique values in the species.
train_data['species'].nunique()
```

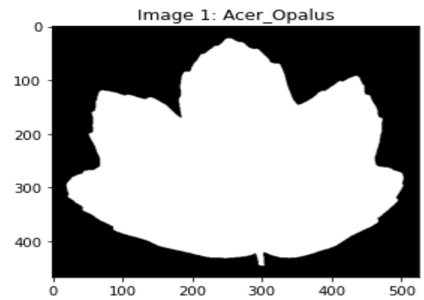
```
99
```

```
#see value counts of every unique value in species.
train_data['species'].value_counts()
```

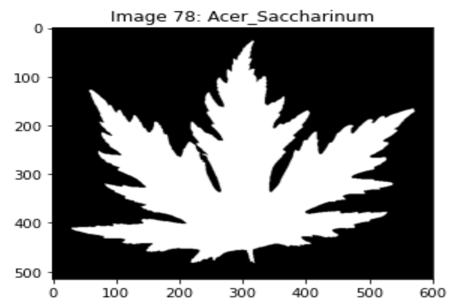
```
Acer_Opalus                    10
Crataegus_Monogyna             10
Acer_Mono                      10
Magnolia_Heptapeta             10
Acer_Capillipes                10
                               ..
Alnus_Rubra                    10
Rhododendron_x_Russellianum    10
Cytisus_Battandieri            10
Liriodendron_Tulipifera        10
Sorbus_Aria                    10
Name: species, Length: 99, dtype: int64
```

4. Visualize some images:
   Draw a specific image using its id by imshow and imread functions.

```
#draw image with id=1.
id = 1
plt.imshow(mpimg.imread("images/" + str(id) + ".jpg"), cmap="gray")
plt.title("Image " + str(id) + ": " + train_data[train_data["id"] == id].values[:,1][0]);
```

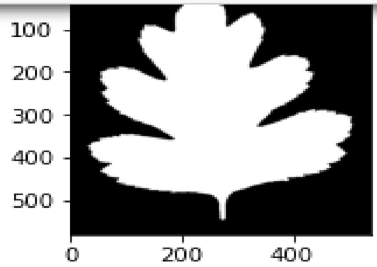


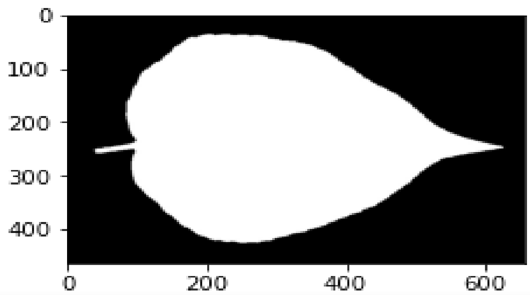

```
#draw image with id=78.
id = 78
plt.imshow(mpimg.imread("images/" + str(id) + ".jpg"), cmap="gray")
plt.title("Image " + str(id) + ": " + train_data[train_data["id"] == id].values[:,1][0]);
```



Drawing some random images from the images file directly using the load_img function.

```
#draw some images randomly from the images file.
from tensorflow.keras.utils import load_img
for i in range(30):
    fig = plt.figure(figsize=(40,30))
    a=fig.add_subplot(10,3,i+1)
    j=np.random.choice((os.listdir('images')))
    img=load_img(os.path.join('images',j))
    plt.imshow(img)
```





5. Correlation analysis:
   Drawing the correlation matrix to see the correlation between features.

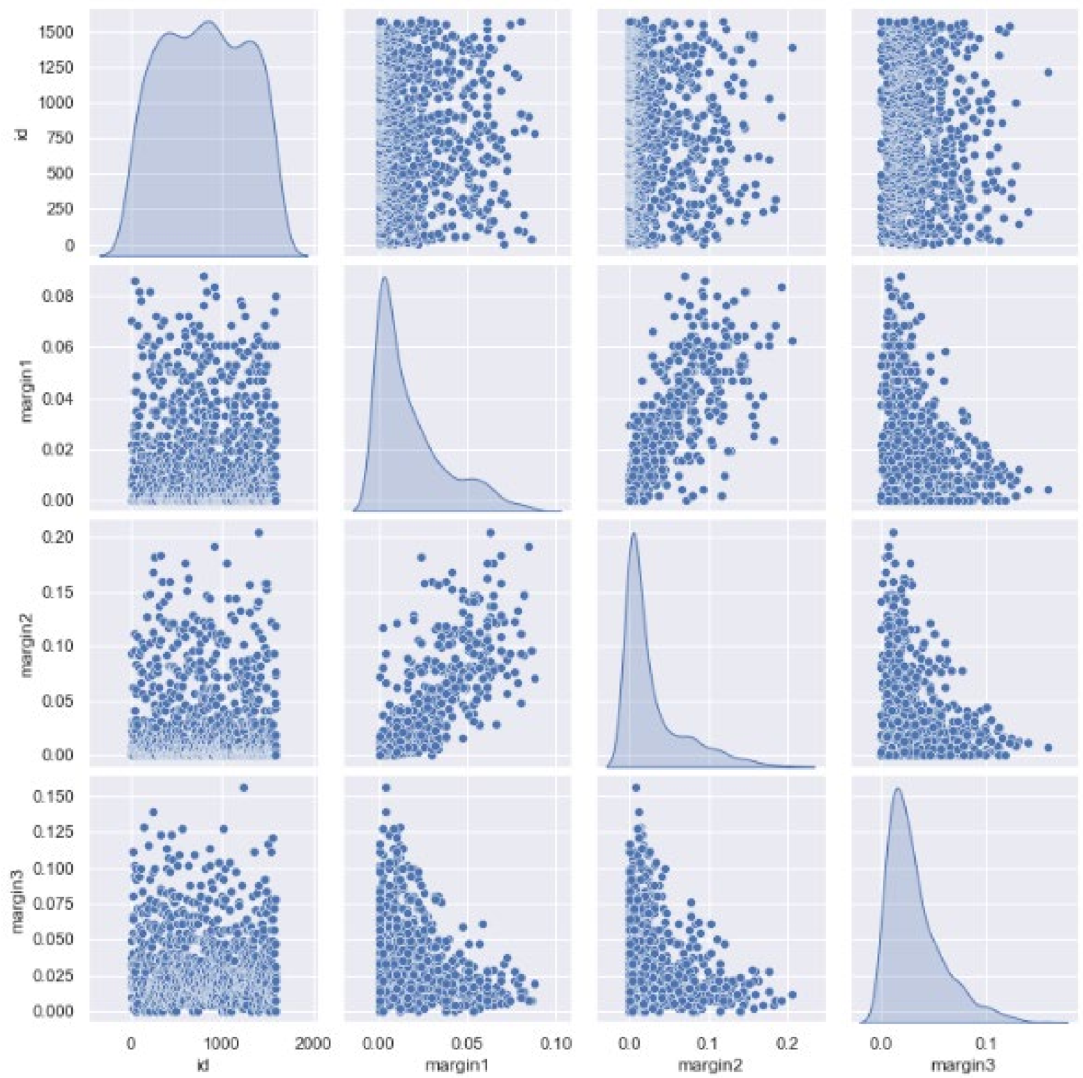
```
# draw the colleration map to see the correlation between features.
corr_map = train_data.corr()
corr_map.style.background_gradient(cmap="Purples")
```

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	margin11	margin12	margin13
id	1.000000	-0.011673	-0.027565	-0.059533	0.001639	-0.002419	-0.051818	0.061214	-0.039509	-0.070954	0.016381	0.020884	0.019541	-0.052985
margin1	-0.011673	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
margin2	-0.027565	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
margin3	-0.059533	-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
margin4	0.001639	-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
margin5	-0.002419	-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
margin6	-0.051818	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
margin7	0.061214	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
margin8	-0.039509	-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
margin9	-0.070954	-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
margin10	0.016381	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
margin11	0.020884	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
margin12	0.019541	-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
margin13	-0.052985	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
margin14	-0.044146	-0.370460	-0.316377	0.095701	0.338136	0.095697	-0.317465	-0.357485	0.001100	0.372013	-0.416157	-0.373834	0.207537	-0.274913
margin15	0.013458	-0.540974	-0.503059	0.050113	-0.259813	0.467991	-0.489144	0.004146	0.062293	-0.117375	-0.320361	-0.603805	0.806811	-0.441614
margin16	0.072274	-0.072127	-0.068356	-0.054076	-0.021615	0.081766	-0.065768	-0.023989	0.205817	-0.026071	-0.053492	-0.081410	0.061776	-0.068353
margin17	0.020472	0.316704	0.135000	-0.130220	-0.047704	-0.235063	0.120157	0.396388	0.025698	-0.236991	0.566792	0.226382	-0.340929	0.006105

6. Data visualization:
   Some visualizations of data are added to see the distributions and understand them better.

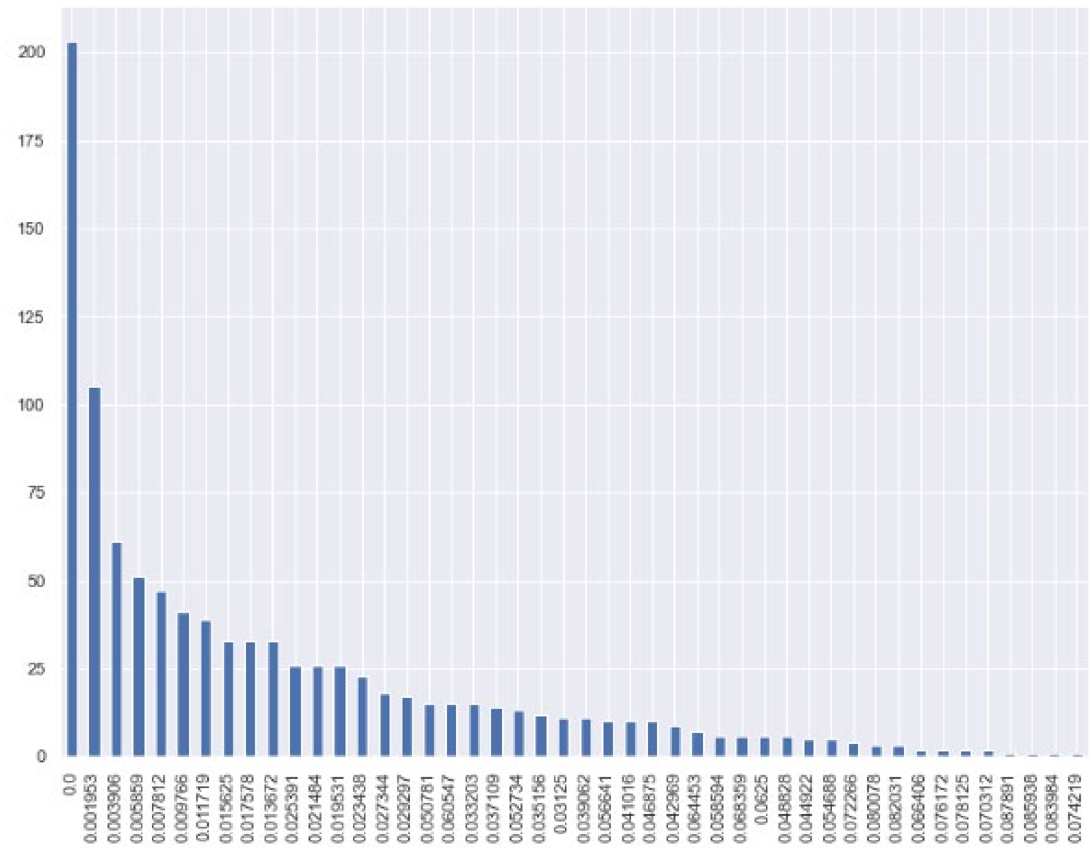
```
#to see some distribution of the features with each other.
sns.set(rc={'figure.figsize':(12,9)})
cData_attr = train_data.iloc[:, 0:5]
sns.pairplot(cData_attr, diag_kind='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x1f457a1aac0>
```



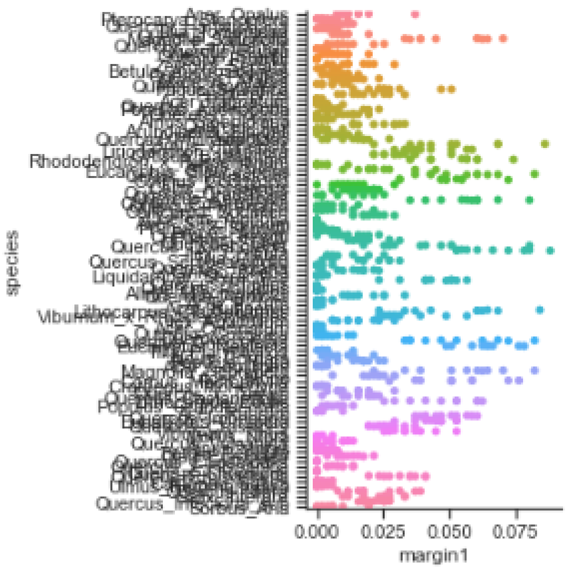
```python
#see the margin 1 column bar plot of its values
train_data["margin1"].value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```




```python
# margin1 w.r.t species
sns.set_theme(style="ticks", color_codes=True)

sns.catplot(x="margin1", y="species", data=train_data)
```

```
<seaborn.axisgrid.FacetGrid at 0x12b4f896b80>
```



7. Label encoding, normalize and train/test split:

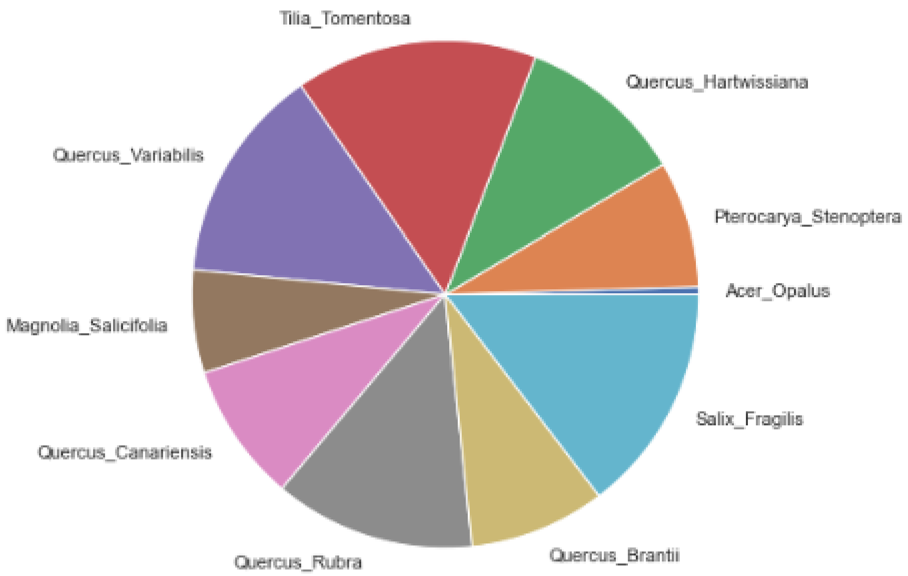
```
X = train_data.drop(["id", "species"], axis = 1, inplace = False).values
encoder = preprocessing.LabelEncoder()
y = encoder.fit_transform(train_data["species"].values)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.2, random_state = 42)
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Visualize labels after label encoder.

```
y_train
```

```
array([17,  8, 69,  9,  3, 30, 56, 86, 33, 82, 41, 34, 49, 21, 88, 27, 29,
        6, 96, 47,  4, 86, 94, 83, 10, 57, 50,  8, 70, 47, 13, 64, 37, 51,
       42, 17, 10, 65, 54, 48,  9, 25, 80, 44, 40, 83, 58,  6, 80, 62, 53,
       79, 73, 37,  2,  1, 76, 95, 35, 86,  7, 68, 83,  0, 85, 39, 37, 92,
       23, 89, 98, 43, 18, 46, 50, 97, 21, 14, 35, 72, 85,  3, 60, 27, 43,
       98, 49, 27, 35, 40, 36,  6, 78, 43, 47, 89,  7, 86, 95,  9,  1, 73,
       38, 11, 68, 19, 18, 41, 72, 78, 71, 77, 25, 11, 83,  1, 95, 65, 79,
       15, 36, 93, 30, 59,  5, 61,  3, 51, 72, 96, 56,  8, 16, 54, 41, 87,
       58, 96, 65, 59, 48, 83, 15, 63, 84, 77, 45, 12, 42, 98,  9, 57, 76,
       38, 68,  2, 73,  2, 76, 17, 10, 76, 66, 74, 87, 54,  2, 50, 60, 65,
       82, 46, 40, 58, 51, 13, 95, 47,  7, 81, 64,  4, 65,  4, 15, 88, 58,
       26, 26, 11, 34, 15, 39, 69, 13, 16, 17, 97, 47, 71, 12, 34, 94, 82,
       48, 65, 23, 82, 86, 78, 75, 97, 91, 10, 90, 59,  5, 88, 34, 74, 24,
       50, 21, 96, 94, 74, 81, 29, 88, 75, 13,  3, 25, 29, 57, 68, 92, 30,
       90, 47, 57, 24,  5, 16, 93, 29, 19, 55, 38, 91, 71,  4, 69, 79,  8,
       46, 51, 64, 95, 16, 56,  0,  1, 25, 14,  6, 92, 50, 87, 84,  9, 16,
        8, 39, 58, 69, 48, 96, 62, 70, 24, 11, 32,  7, 12,  7, 90, 73, 91,
       42,  5, 94, 81, 12, 92, 58, 86,  2, 96, 38, 74, 21, 19, 31, 94, 70,
       45, 19, 21, 91, 24,  5, 71, 14, 77, 62, 85, 13, 29, 77, 67, 44, 88,
       40, 77, 86, 18,  1, 28,  4, 78, 93, 87,  8, 66, 72, 46, 70, 84, 22,
       14, 95, 36, 19, 54, 77, 23,  0, 94, 19, 13, 21, 57, 52, 43, 63, 18,
       27, 31, 50, 48, 85, 31, 74, 92,  0, 45, 90, 50, 25, 34, 48, 61, 97,
       20, 55, 44, 11, 90, 44, 67, 64, 63, 90, 55, 73, 20, 61, 15, 56, 55,
       90, 54,  1, 59, 79, 88, 18,  4, 55, 86, 60, 39, 59, 71, 51, 18,  3,
       53, 85, 27,  7,  4, 55, 96, 70, 76, 97, 23, 70, 75, 43, 55, 54, 89,
       43, 25, 19, 63, 15, 58, 89, 87, 73,  7, 14,  6, 33, 92, 25, 85, 66,
       18, 32, 42, 85, 48, 14, 88,  4, 67, 59, 32, 83,  5, 57, 12, 30, 89,
       72, 69,  0, 18, 76, 10, 84, 92, 94, 42, 79, 62, 81, 67, 22, 42, 71,
       93, 68,  7, 53, 81, 21, 91, 93, 15, 67, 68, 33, 69, 78, 38, 95, 26,
       63, 28, 71, 68,  6, 49, 11, 22, 36, 31, 60, 87, 57, 92,  4,  8, 23,
       32, 37, 35, 35, 33, 37, 45, 40, 49, 76, 70, 74, 26,  6, 63, 98, 32,
       82, 33, 39, 79, 77, 38, 66, 97, 64, 64,  3, 60, 17, 69, 95, 20, 22,
        0, 49, 96, 46, 94, 82, 83, 75, 36, 11, 69, 62, 78,  0,  1, 26, 38,
       82, 77, 41, 93, 30, 37, 71, 43, 53, 98, 19, 36, 98, 44, 45, 24, 89,
       58, 80, 86, 47, 60,  0, 55, 79, 75, 72, 41, 41, 68,  1,  2, 75,  8,
       87, 37, 92, 77, 84, 52, 28, 84, 29, 15, 98, 87, 30,  3, 20, 57, 62,
        8,  0, 21, 68,  6, 23, 35, 86, 32, 61,  8, 74, 39, 26, 84, 10, 43,
       80, 44,  9,  7, 61, 74, 29, 31, 23, 76, 56, 37, 52, 94, 42, 33, 16,
       56, 22, 17, 51, 72, 96, 25, 46, 61, 49, 53, 63,  2, 75, 34, 63, 49,
       34, 81, 59, 23, 23, 79, 13, 58, 61, 84, 22, 31, 57, 97,  9,  3, 64,
       22, 27, 53, 50, 64, 16, 14, 27, 69, 60, 16,  5, 32, 67, 80, 42, 46,
       46, 65, 30,  3, 31,  3, 28, 59, 80, 93, 49, 87,  2, 53, 51, 69, 12,
       68, 81, 88, 22, 79,  1, 18, 90, 62, 20,  7,  6, 50, 31, 12, 24, 53,
       12, 34, 70, 12, 52, 13, 56, 81, 94, 38, 56, 77, 56, 54, 39, 14, 90,
       30,  4, 63, 81, 28, 47, 30, 31, 78, 84, 22, 70, 87, 47, 80, 61, 42,
       57, 24, 91, 80, 62, 10, 88,  1, 40, 38, 16, 17, 20, 28, 73, 39, 29,
       24, 24, 83, 62, 35, 84, 42, 29, 91, 93])
```

```
#visualization of labels.
fig = plt.figure(figsize =(10, 7))
plt.pie(y[:10],labels=train_data['species'][:10]);
```



Part II: Training a neural network

1. Model evaluation function:

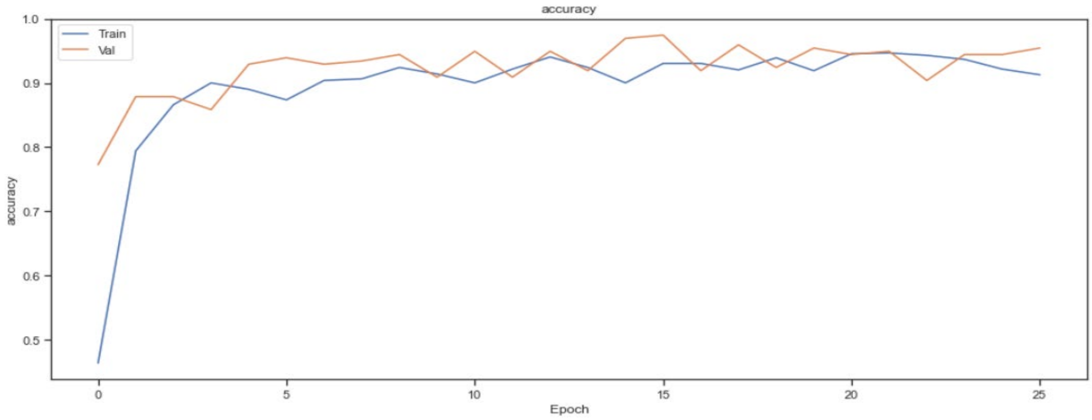
```
# this function take hyperparameters like: batch size, hidden layers, optimizers, dropout, learning_rate and regulazation as inpu
# these hyperparameter takes a defult value in the function so that if any value changed in the call the rest stay the same.
#this function build,compile,fit,evaluate and draw training curves.
def evaluate_diff_model(batch_size = 16, hidden_layer = 384, optimizer = "Adam", dropout = 0.5,
                        learning_rate = 0.01, regularization = 0.01):
    model = tf.keras.Sequential() #to build the model.
    model.add(tf.keras.Input(shape=(192,)))#number of hidden units=number of features.
    model.add(tf.keras.layers.Dense(hidden_layer, activation = "tanh"))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(99, activation = "softmax"))#number of hidden units=number of labels.
    if optimizer == "Adam":
        opt = tf.keras.optimizers.Adam(learning_rate = learning_rate, weight_decay = regularization)
    elif optimizer == "SGD":
        opt = tf.keras.optimizers.SGD(learning_rate = learning_rate, weight_decay = regularization)
    elif optimizer == "RMSprop":
        opt = tf.keras.optimizers.RMSprop(learning_rate = learning_rate, weight_decay = regularization)
    else:
        print("Invalid Optimizer Name")
        return
    model.compile(optimizer = opt, loss = "SparseCategoricalCrossentropy", metrics = ["accuracy"])# to compile the model.
    EarlyStop = tf.keras.callbacks.EarlyStopping(patience = 10)

    history = model.fit(X_train, y_train, batch_size = batch_size ,epochs = 1000, verbose = True,
                        validation_data = (X_test, y_test), callbacks=[EarlyStop]) #fit the model.
    print("model evaluation on test data")
    model.evaluate(X_test, y_test);#evaluate the model.

    fig, axes = plt.subplots(2,1, figsize = [16, 16])
    axes[0].plot(history.history['accuracy'])
    try:
        axes[0].plot(history.history['val_accuracy'])
        axes[0].legend(['Train', 'Val'])
    except:
        pass
    axes[0].set_title('{:s}'.format('accuracy'))
    axes[0].set_ylabel('{:s}'.format('accuracy'))
    axes[0].set_xlabel('Epoch')
    fig.subplots_adjust(hspace=0.5)
    axes[1].plot(history.history['loss'])
    try:
        axes[1].plot(history.history['val_loss'])
        axes[1].legend(['Train', 'Val'])
    except:
        pass
    axes[1].set_title('Model Loss')
    axes[1].set_ylabel('Loss')
    axes[1].set_xlabel('Epoch')
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
```

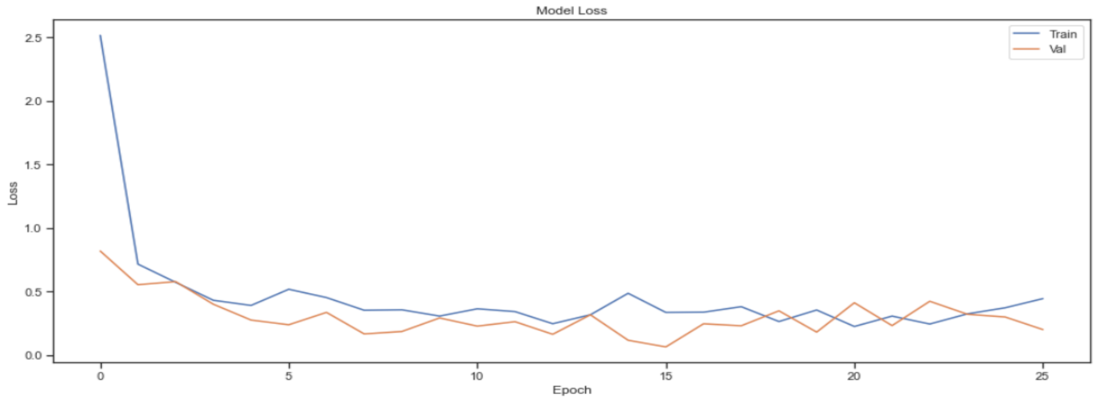
2. Train models with different hyperparameters:
   The model will be trained with different hyperparameters.

Hyperparameter 1 ’optimizers’:

I. with Adam optimizer
   model evaluation with test data:
   loss: 0.1991 - Accuracy: 0.9545

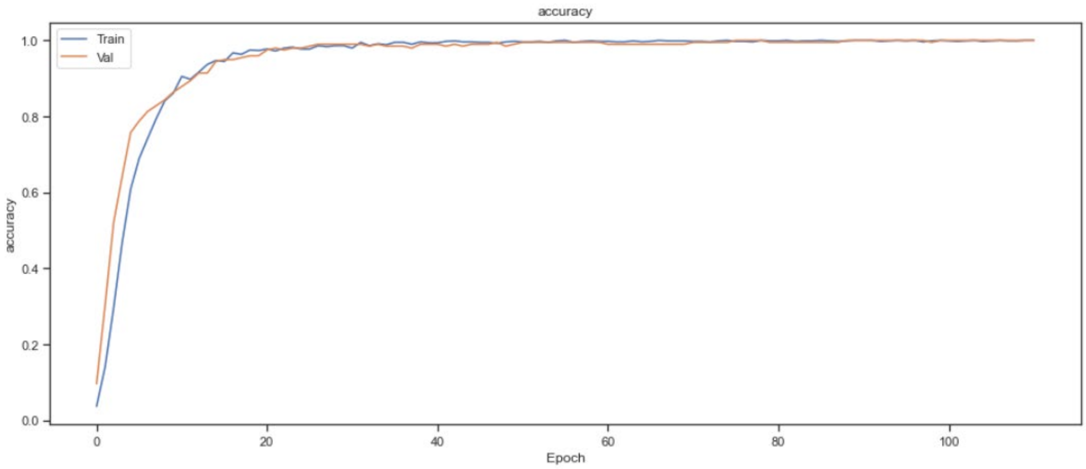
model accuracy and loss curves:

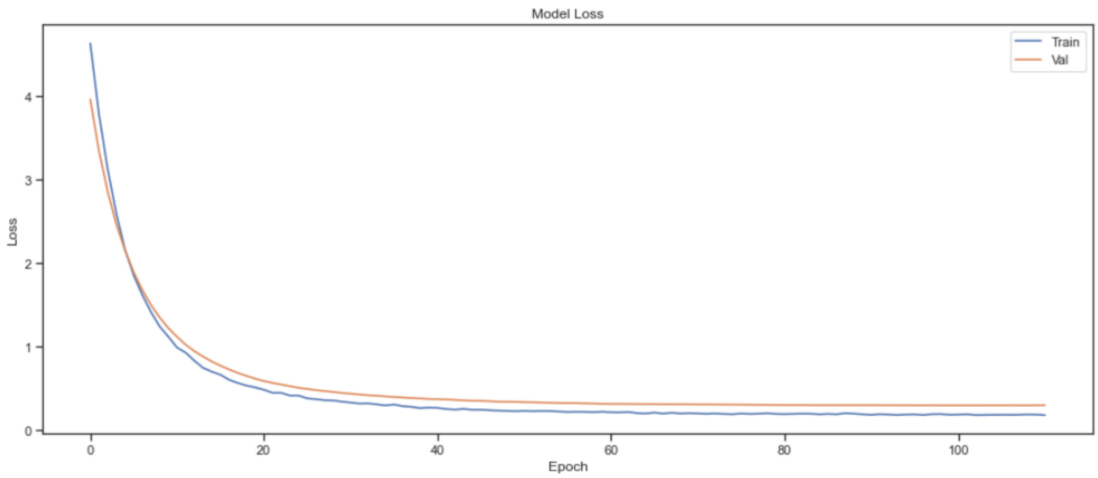




II. with SGD optimizer
model evaluation with test data:
loss: 0.3004 - Accuracy: 1.0000

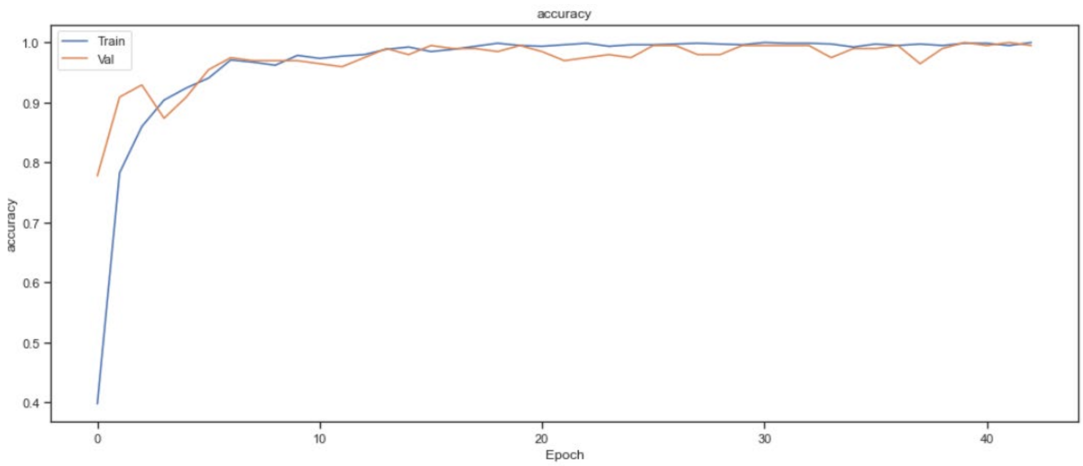
model accuracy and loss curves:

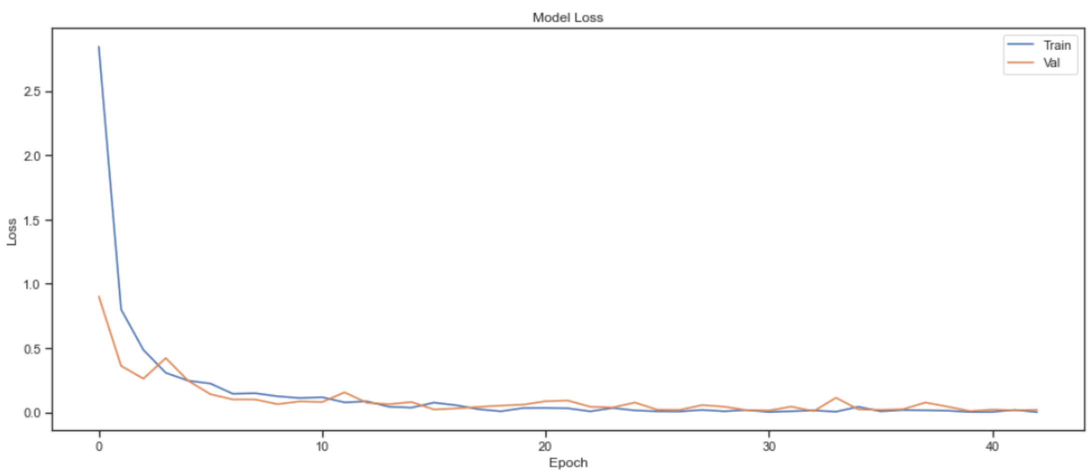




III. with RMSprop optimizer
model evaluation with test data:
loss: 0.0185 - Accuracy: 0.9949

model accuracy and loss curves:





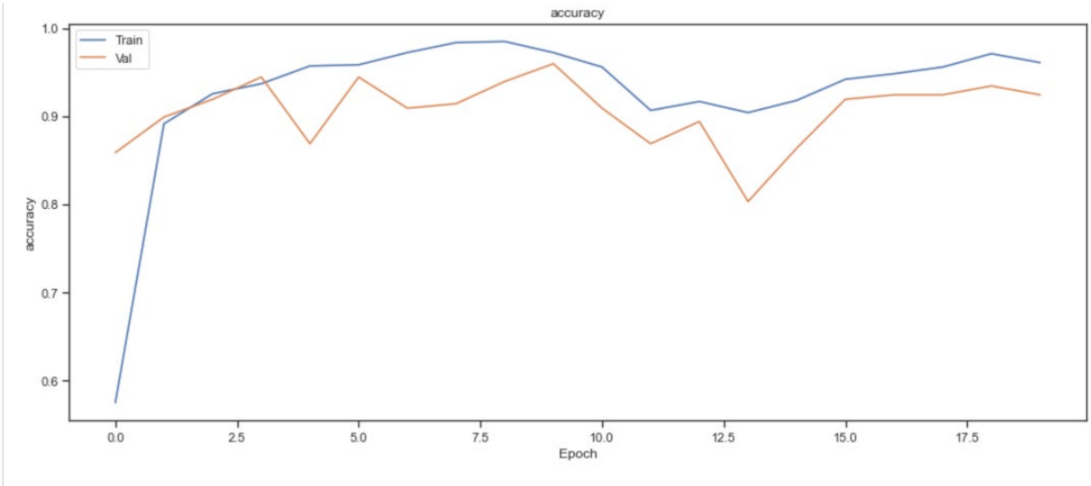
Hyperparameter 2 ‘dropout’:

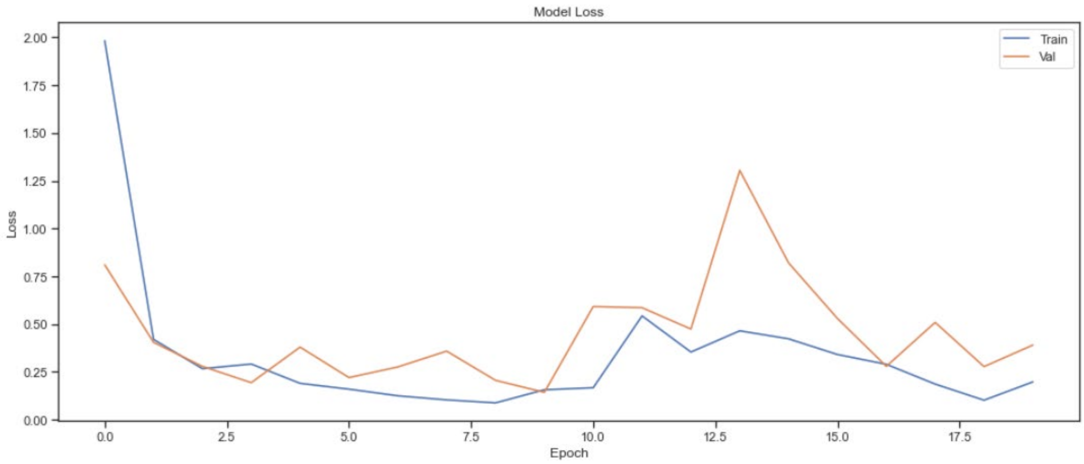
a. dropout = 0.2

model evaluation with test data:

loss: 0.3896 - Accuracy: 0.9242

model accuracy and loss curves:



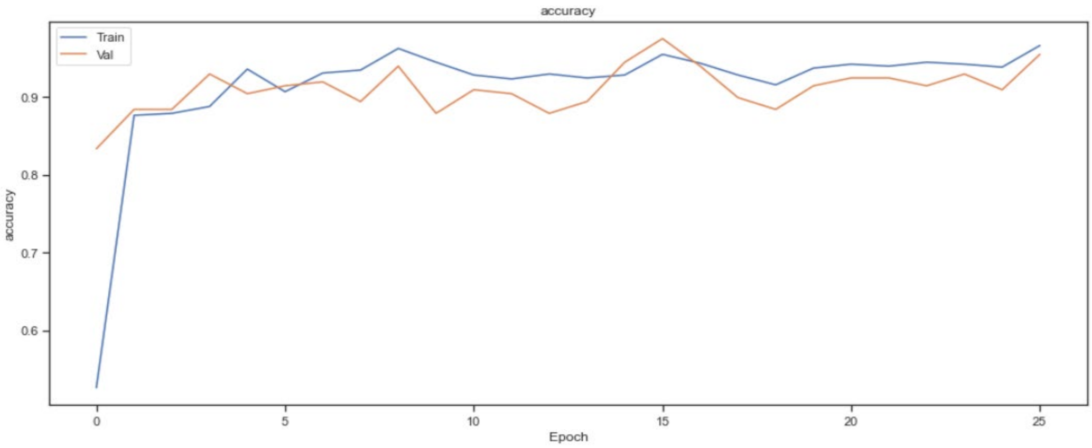


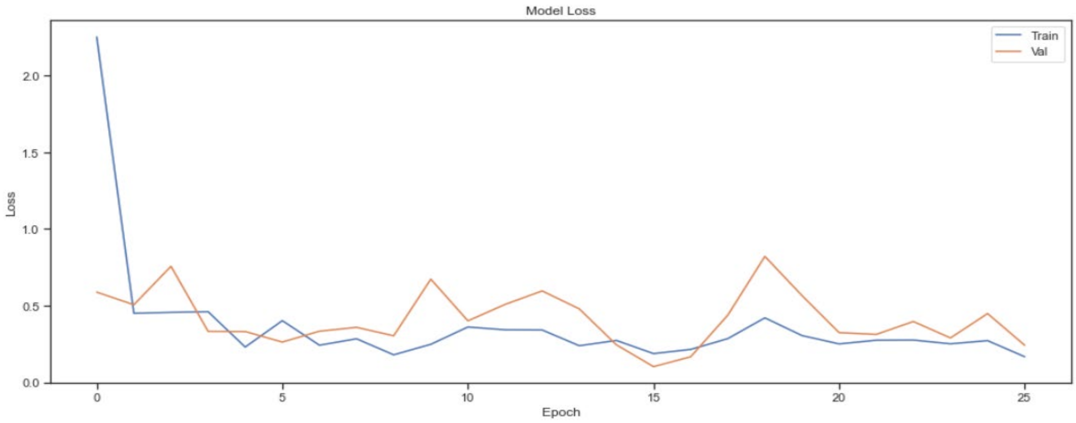
b. dropout=0.4

model evaluation with test data:

loss: 0.2421 - Accuracy: 0.9545

model accuracy and loss curves:



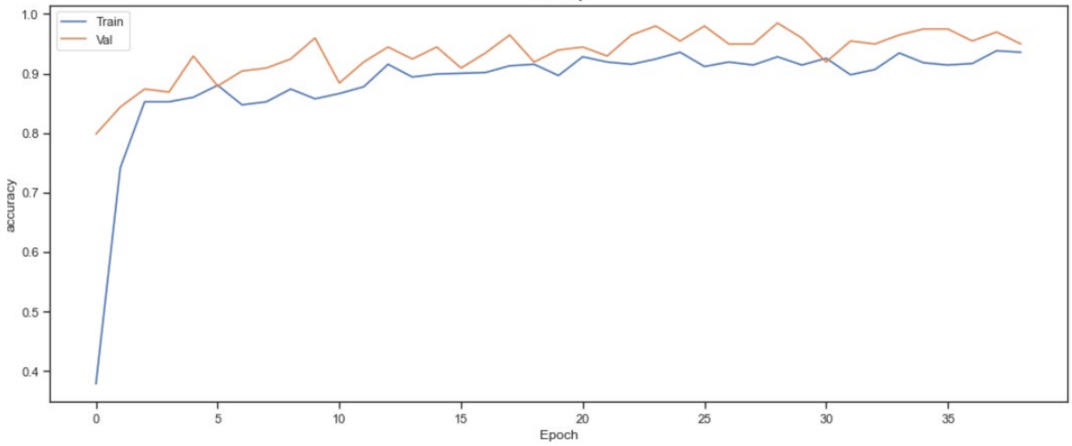


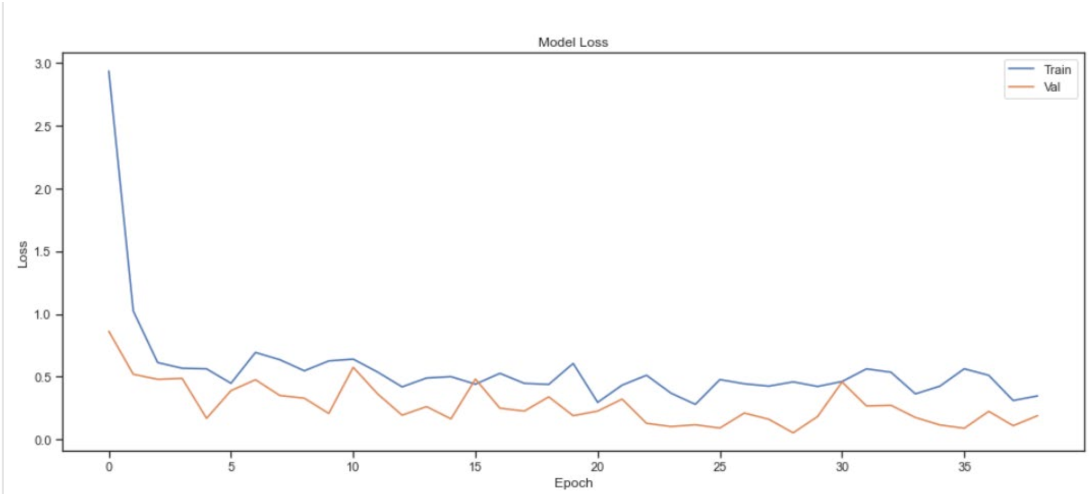
c. dropout=0.6

model evaluation with test data:

loss: 0.1902 - Accuracy: 0.9495

model accuracy and loss curves:





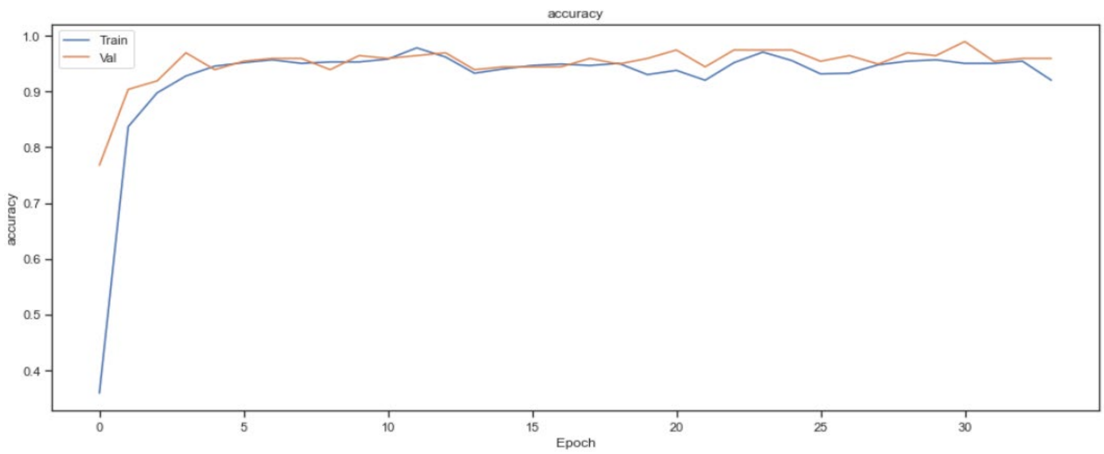
Hyperparameter 3 ‘hidden layer units’:

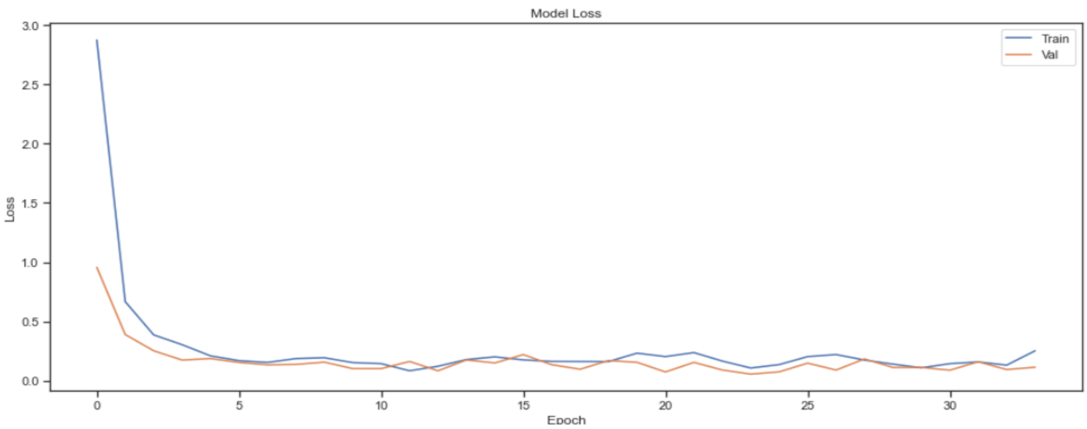
a. hidden_layer = 128

model evaluation with test data:

loss: 0.1147 - Accuracy: 0.9596

model accuracy and loss curves:



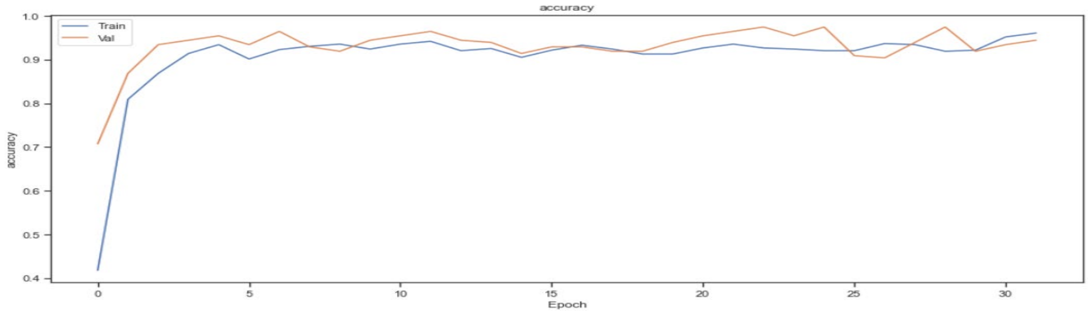


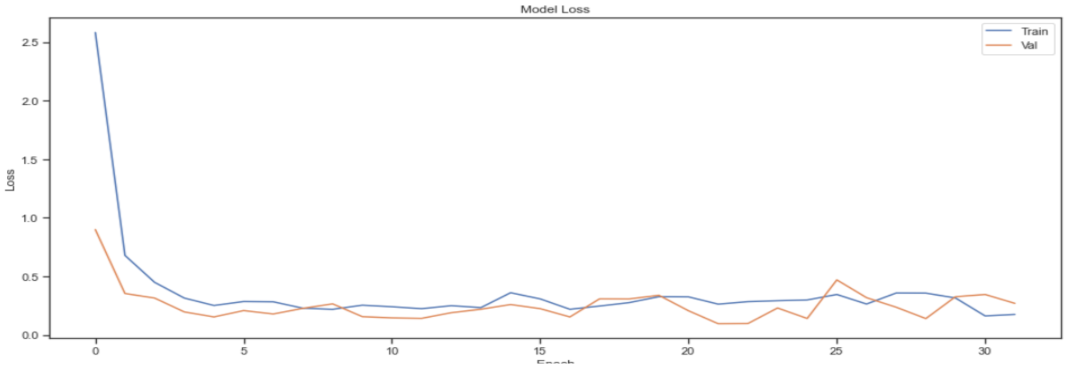
b. hidden_layer = 256

model evaluation with test data:

loss: 0.2702 - Accuracy: 0.9444

model accuracy and loss curves:



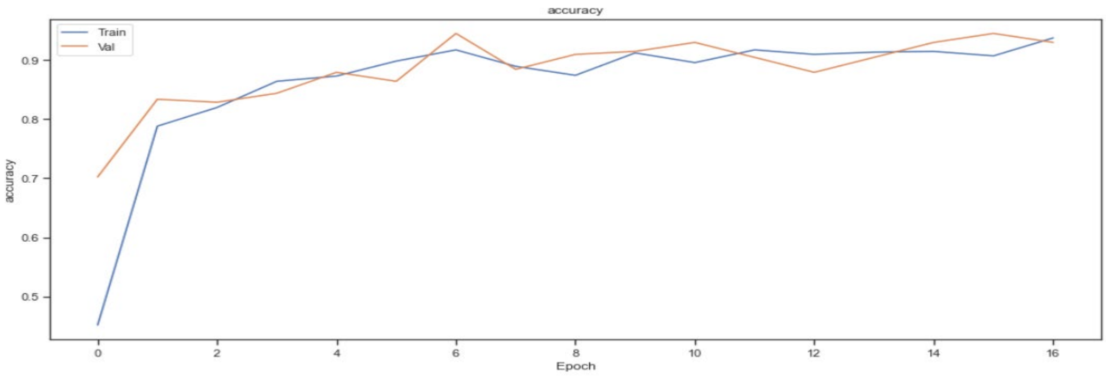


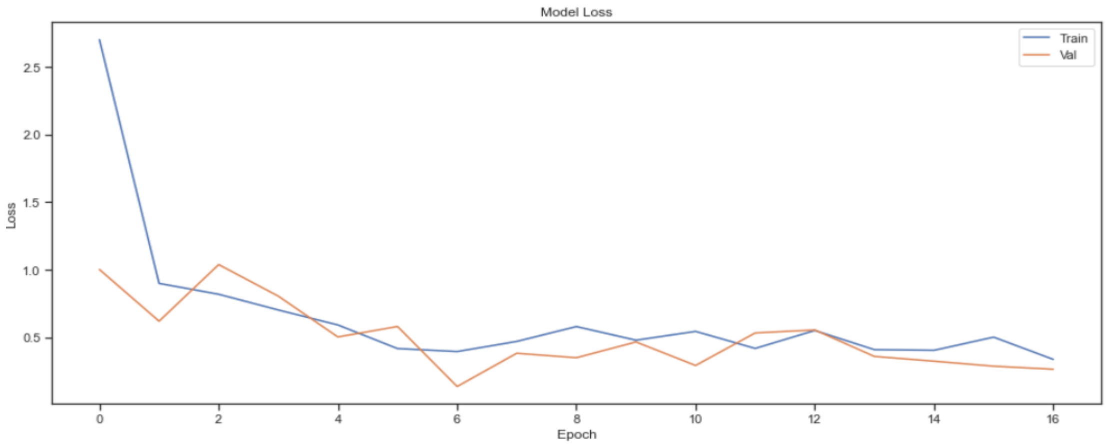
c. hidden_layer = 512

model evaluation with test data:

loss: 0.2639 - Accuracy: 0.9293

model accuracy and loss curves:





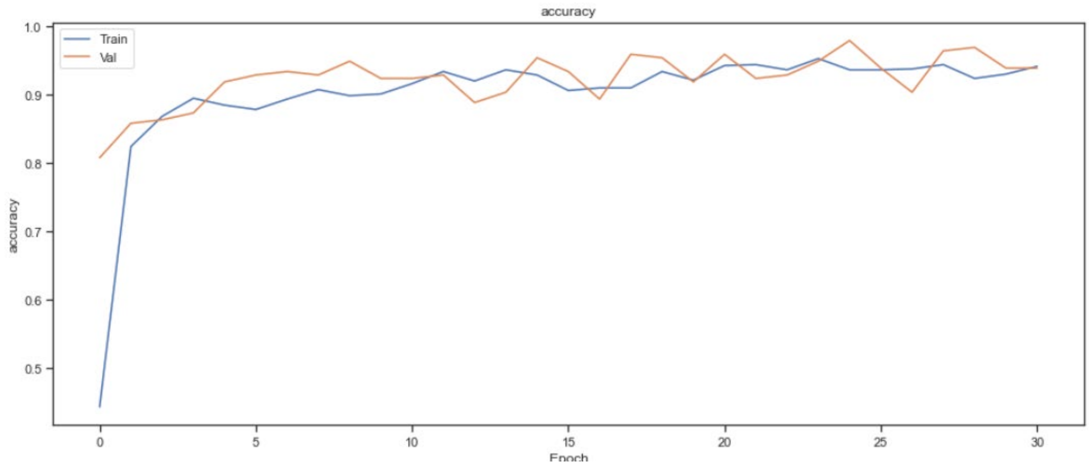
Hyperparameter 4 ‘batch size’:

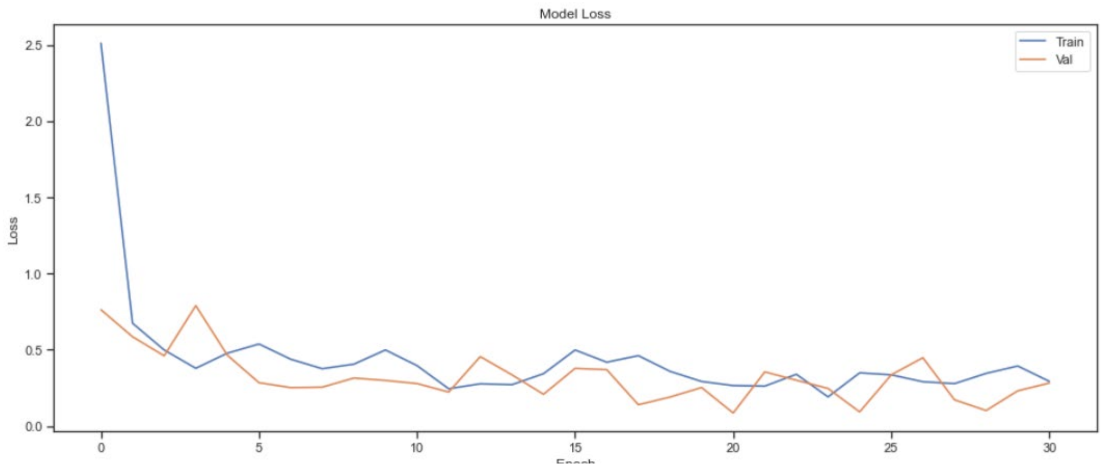
a. batch_size=16

model evaluation with test data:

loss: 0.2810 - Accuracy: 0.9394

model accuracy and loss curves:



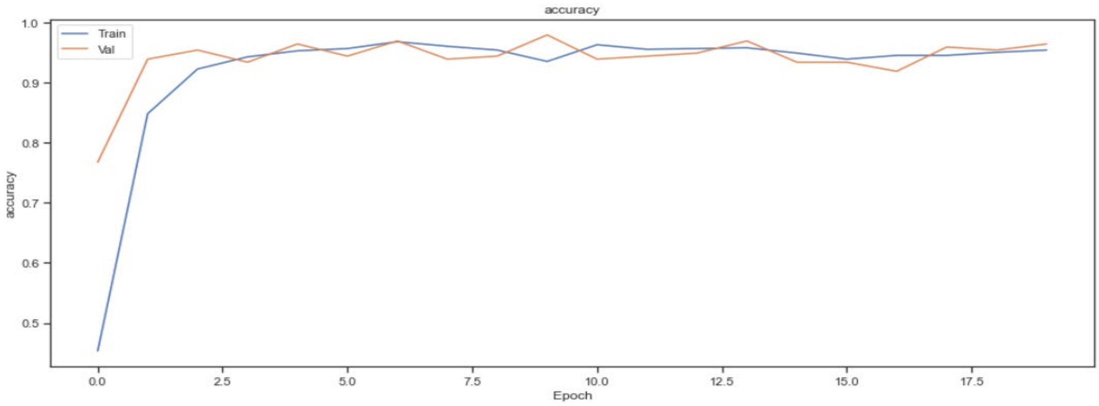


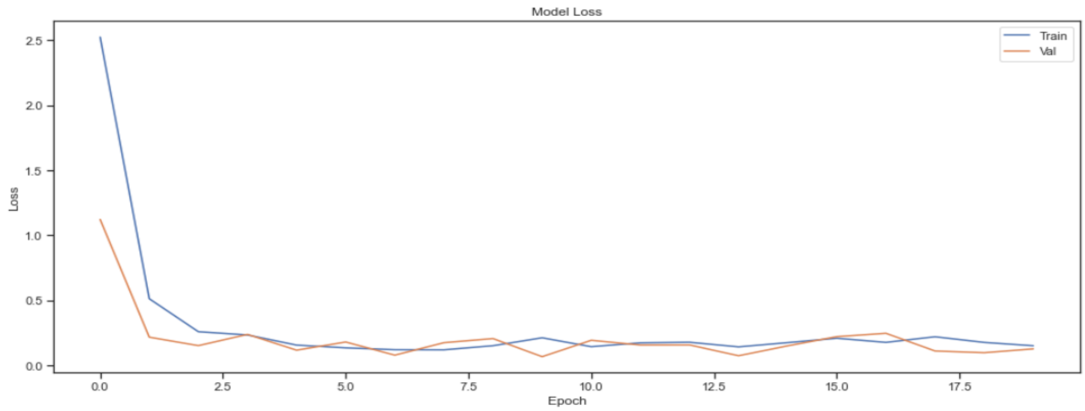
b. batch_size=32

model evaluation with test data:

loss: 0.1270 - Accuracy: 0.9646

model accuracy and loss curves:



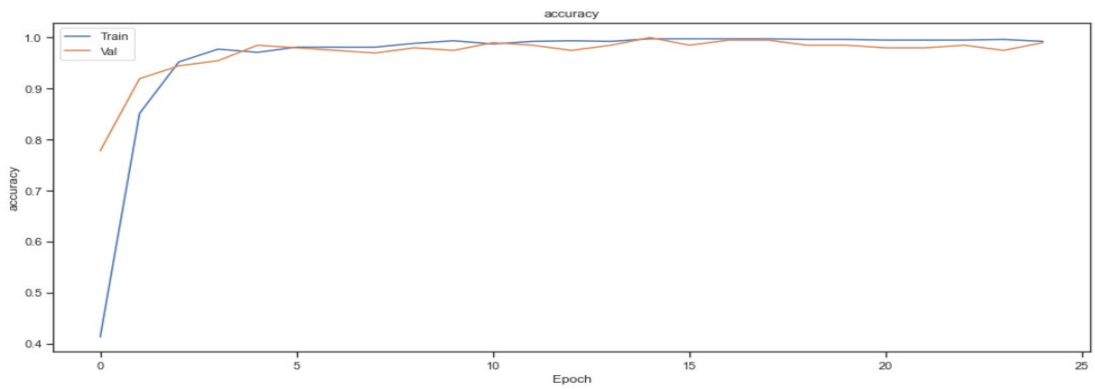


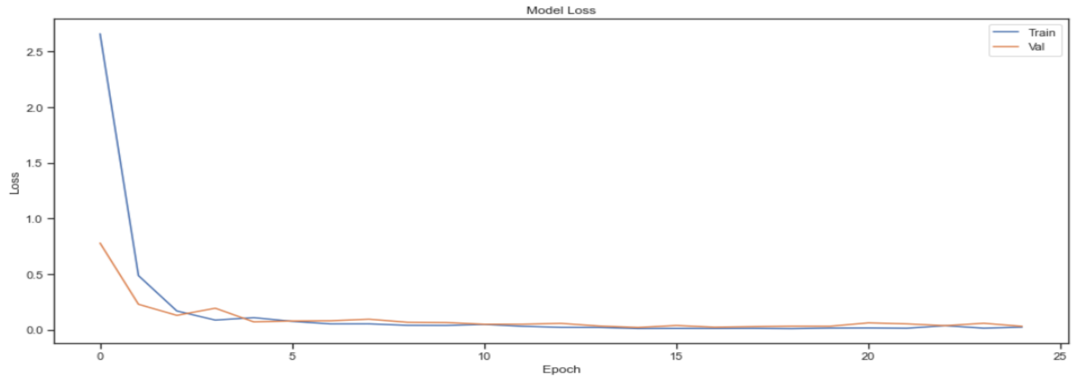
c. batch_size=64

model evaluation with test data:

loss: 0.0312 - Accuracy: 0.9899

model accuracy and loss curves:





The best model:

Using hyperparameters:

- optimizer="RMSprop"
- dropout=0.4
- hidden_layer =128
- batch_size = 64

model evaluation with test data:

loss: 0.0169 - Accuracy: 0.9899

model accuracy and loss curves: