Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024
ACTIVITY	**Assignment 8.1**
Name	Cuevas, Christian Jay L.
Section	CPE32S3
Date Performed:	4/14/2024
Date Submitted:	4/19/2024
Instructor:	Engr. Roman M. Richard

Instructions:

1. Choose any dataset applicable to either a classification problem or a regression problem.
2. Explain your datasets and the problem being addressed.
3. Show evidence that you can do the following:
   - Save a model in HDF5 format
   - Save a model and load the model in a JSON format
   - Save a model and load the model in a YAML format
   - Checkpoint Neural Network Model Improvements
   - Checkpoint Best Neural Network Model only
   - Load a saved Neural Network model
   - Visualize Model Training History in Keras
   - Show the application of Dropout Regularization
   - Show the application of Dropout on the visible layer
   - Show the application of Dropout on the hidden layer
   - Show the application of a time-based learning rate schedule
   - Show the application of a drop-based learning rate schedule

Datasets

Classification Task

- I will be using User Knowledge Dataset for Classification task.
- For the classification task, the User Knowledge Modeling Dataset consists of 6 features, 5 predictors and 1 target variable.
- The problem that is being solved here is to predict the knowledge level of the user about DC Electrical machine just by using the data provided through online web-courses like the study time and their exam performance. Through this, we can estimate the level of learners and observe if they are really learning or not. We can also use this to gauge the feasibility of the online courses.
- Below are the definitions of the features of this dataset:
  - STG (The degree of study time for goal object materails), (input value)
  - SCG (The degree of repetition number of user for goal object materails) (input value)
  - STR (The degree of study time of user for related objects with goal object) (input value)
  - LPR (The exam performance of user for related objects with goal object) (input value)
  - PEG (The exam performance of user for goal objects) (input value)
  - UNS (The knowledge level of user) (target value)

- **These are the values for the class labels:**
  - **Very Low: 50**
  - **Low:129**
  - **Middle: 122**
  - **High 130**

(All of the information are from the authors of this dataset.)

Importing Libraries and Dataset

In [1]:

```
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

In [2]:

```
!pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.13.0-py3-none-any.whl (26 kB)
Collecting keras>=3.2.0 (from scikeras)
  Downloading keras-3.2.1-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 10.2 MB/s eta 0:00:00
Collecting scikit-learn>=1.4.2 (from scikeras)
  Downloading scikit_learn-1.4.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
 (12.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 31.6 MB/s eta 0:00:00
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from ker
as>=3.2.0->scikeras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras
>=3.2.0->scikeras) (1.25.2)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>
=3.2.0->scikeras) (13.7.1)
Collecting namex (from keras>=3.2.0->scikeras)
  Downloading namex-0.0.8-py3-none-any.whl (5.8 kB)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras>
=3.2.0->scikeras) (3.9.0)
Collecting optree (from keras>=3.2.0->scikeras)
  Downloading optree-0.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (311
kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.2/311.2 kB 34.9 MB/s eta 0:00:00
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from k
eras>=3.2.0->scikeras) (0.2.0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (fro
m scikit-learn>=1.4.2->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (fr
om scikit-learn>=1.4.2->scikeras) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packa
ges (from scikit-learn>=1.4.2->scikeras) (3.4.0)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.10/dist-p
ackages (from optree->keras>=3.2.0->scikeras) (4.11.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-pack
ages (from rich->keras>=3.2.0->scikeras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-pa
ckages (from rich->keras>=3.2.0->scikeras) (2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from
markdown-it-py>=2.2.0->rich->keras>=3.2.0->scikeras) (0.1.2)
Installing collected packages: namex, optree, scikit-learn, keras, scikeras
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
  Attempting uninstall: keras
    Found existing installation: keras 2.15.0
    Uninstalling keras-2.15.0:
      Successfully uninstalled keras-2.15.0
```

```
ERROR: pip's dependency resolver does not currently take into account all the packages that
are installed. This behaviour is the source of the following dependency conflicts.
tensorflow 2.15.0 requires keras<2.16,>=2.15.0, but you have keras 3.2.1 which is incompati
ble.
Successfully installed keras-3.2.1 namex-0.0.8 optree-0.11.0 scikeras-0.13.0 scikit-learn-1
.4.2
```

In [83]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from keras.optimizers import Adam, SGD, RMSprop
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Activation, Dropout
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_c
urve, accuracy_score
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from scikeras.wrappers import KerasRegressor
from sklearn.pipeline import Pipeline
from tensorflow.keras.models import model_from_json
import os
from keras.models import load_model
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import LearningRateScheduler
import math
```

In [4]:

```
from ucimlrepo import fetch_ucirepo

"""Classification Dataset"""
user_knowledge_modeling = fetch_ucirepo(id=257)

knowledgeDF = user_knowledge_modeling.data.original
```

CLASSIFICATION TASK

Performing Exploratory Data Analysis on Classification Dataset

- **Let's perform Exploratory Data Analysis on the User Knowledge Dataset to better understand and analyze what we're modeling to. EDA is also recommended before making assumptions to the dataset, this is to make it easier for us to spot patterns or outliers in the data [1].**
- **It is good to start first by looking at the .info() of the dataset, to see the data types and the overview of the features.**

In [5]:

```
knowledgeDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 403 entries, 0 to 402
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   STG     403 non-null    float64
 1   SCG     403 non-null    float64
 2   STR     403 non-null    float64
```

```
 3   LPR     403 non-null    float64
 4   PEG     403 non-null    float64
 5   UNS     403 non-null    object
dtypes: float64(5), object(1)
memory usage: 19.0+ KB
```

Observation:

- **As we can see right here, there are 403 data entries and there are in total of 6 features. There are 5 predictors and their data types are all float64. The target variable is object data type. With pure observation, I can see that there are no missing values, but we will further confirm it using the .isna().**

In [6]:

```
knowledgeDF.isna().sum()
```

Out[6]:

```
STG    0
SCG    0
STR    0
LPR    0
PEG    0
UNS    0
dtype: int64
```

Observation:

- **We confirmed that there are no missing or null values in this dataset.**

In [7]:

```
knowledgeDF.head()
```

Out[7]:

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	very_low
1	0.08	0.08	0.10	0.24	0.90	High
2	0.06	0.06	0.05	0.25	0.33	Low
3	0.10	0.10	0.15	0.65	0.30	Middle
4	0.08	0.08	0.08	0.98	0.24	Low

Observation:

- **By using .head(), we got a better overview of the rows and columns, and their corresponding values. We can see that in the first five rows, the values for the predictors are very small and they are in decimal.**

In [8]:

```
knowledgeDF.describe()
```

Out[8]:

	STG	SCG	STR	LPR	PEG
count	403.000000	403.000000	403.000000	403.000000	403.000000
mean	0.353141	0.355940	0.457655	0.431342	0.456360
std	0.212018	0.215531	0.246684	0.257545	0.266775
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.200000	0.200000	0.265000	0.250000	0.250000
50%	0.300000	0.300000	0.440000	0.330000	0.400000
75%	0.480000	0.510000	0.680000	0.650000	0.660000
max	0.990000	0.900000	0.950000	0.990000	0.990000

Observation:

- **We can see the values of the count, mean, std, min, 25%, 50%, 75% and max. By looking at these values, we can infer important information such as the number of entries per column, the dispersion of the data and if there are any outliers. We can see that the std is close to the mean, which means that the data is clustered, also we can observe that the min and max of each column is in the range of 0 - 0.99 and it does not have an outlier with a large gap in value.**

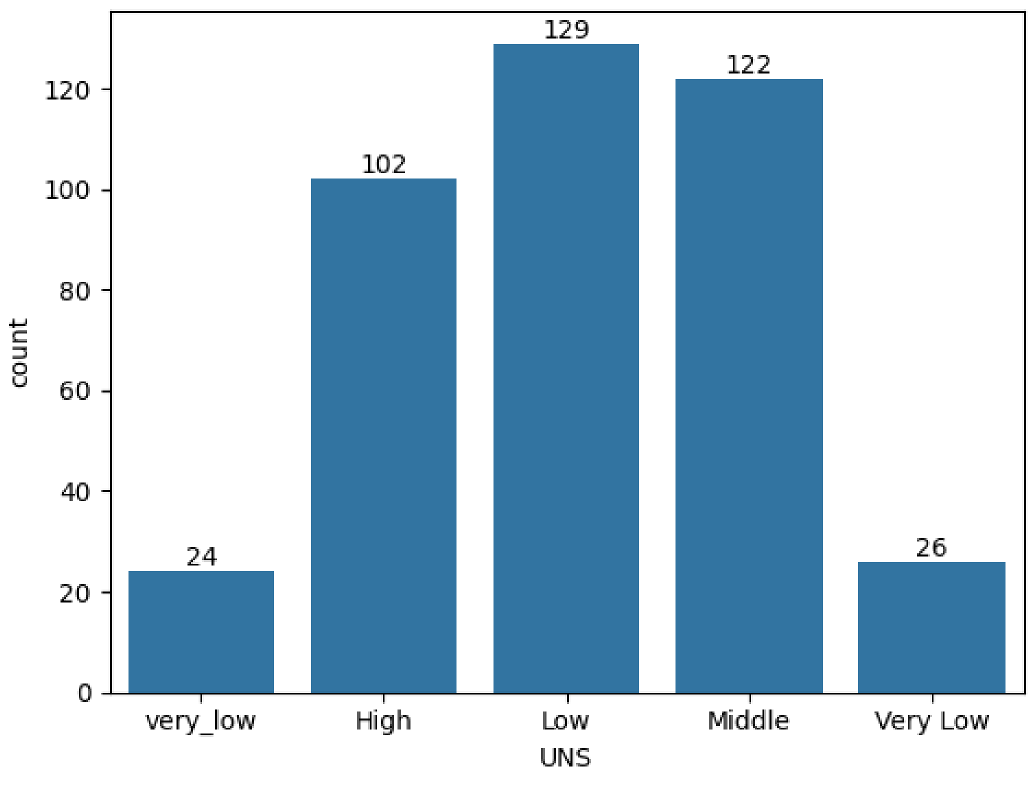
- **We can use lambda function to turn categorical values to integers, specifically ranging from 1-4. 1 is for very low, 2 is for low, 3 is for middle, and 4 is for high.**

In [9]:

```
ax = sns.countplot(data=knowledgeDF, x='UNS')
ax.bar_label(ax.containers[0])
```

Out[9]:

```
[Text(0, 0, '24'),
 Text(0, 0, '102'),
 Text(0, 0, '129'),
 Text(0, 0, '122'),
 Text(0, 0, '26')]
```



In [10]:

```
cat_values = {"very_low":1, "Very Low":1, "Low":2, "Middle":3, "High":4}
knowledgeDF["UNS"] = knowledgeDF["UNS"].apply(lambda toLabel: cat_values.get(toLabel, 0))
```

In [11]:

```
knowledgeDF.head()
```

Out[11]:

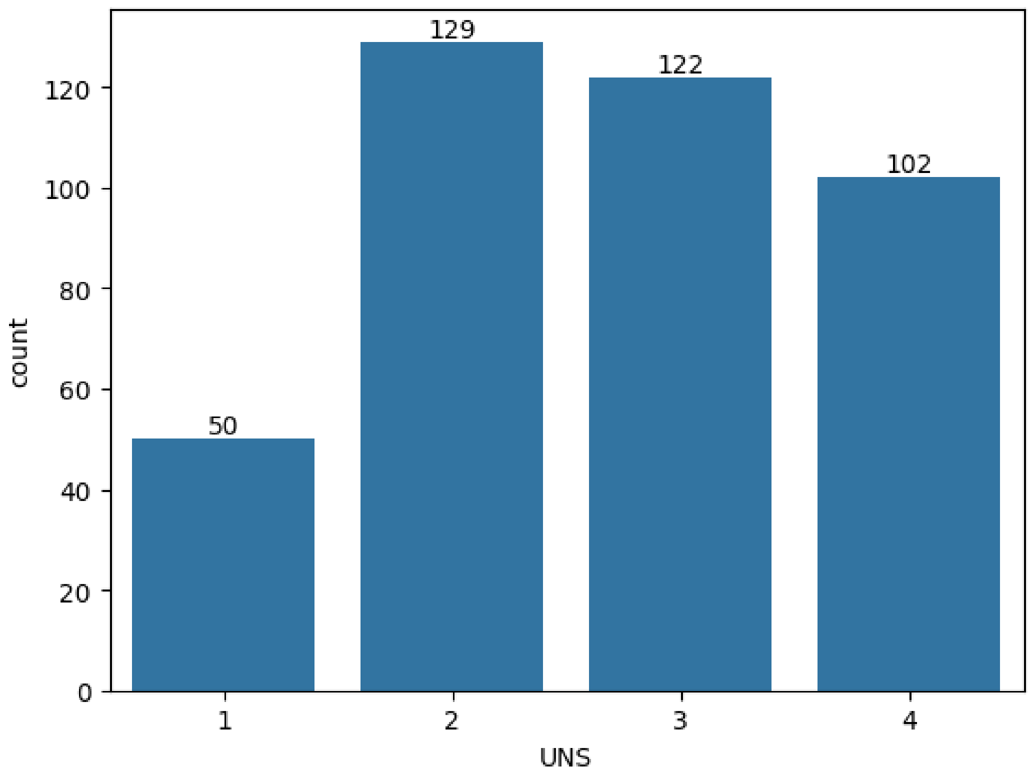
	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	1
1	0.08	0.08	0.10	0.24	0.90	4
2	0.06	0.06	0.05	0.25	0.33	2
3	0.10	0.10	0.15	0.65	0.30	3
4	0.08	0.08	0.08	0.98	0.24	2

In [12]:

```
ax = sns.countplot(data=knowledgeDF, x='UNS')
ax.bar_label(ax.containers[0])
```

Out[12]:

```
[Text(0, 0, '50'), Text(0, 0, '129'), Text(0, 0, '122'), Text(0, 0, '102')]
```



Observation:

- In the first bar graph, we can see that there are 5 labels, they are "very_low", "Low", "Middle", "High", and "Very Low". You will notice that "very_low" and "Very Low" are the same but they were separated in labels because of different naming. I created a dictionary where the "very_low" and "Very Low" have the same value of 1, so when the lambda function convert them to integer, they will be combined together.
- In the second bar graph, we can observe that there are only 4 labels. The "very_low" and "Very Low" were combined into the value 1, the total value of entries with the value of 1 is 50.
- The lambda function has successfully converted the categorical values to integer values. We can now proceed with the correlation analysis and the heatmap visualization to further explore the dataset.

- **We will be looking at the correlation between the target variable and the predictors. We will first look at the table of correlation, then sort the values of correlation with respect to "UNS" and lastly visualize the correlation using heatmap.**

In [13]:

```
knowledgeDF.corr(method = "pearson")
```

Out[13]:

	STG	SCG	STR	LPR	PEG	UNS
STG	1.000000	0.049023	-0.051889	0.113957	0.198629	0.217477
SCG	0.049023	1.000000	0.121235	0.119716	0.193566	0.249095
STR	-0.051889	0.121235	1.000000	0.083423	0.148338	0.203452
LPR	0.113957	0.119716	0.083423	1.000000	-0.039283	0.247968
PEG	0.198629	0.193566	0.148338	-0.039283	1.000000	0.919456
UNS	0.217477	0.249095	0.203452	0.247968	0.919456	1.000000

In [14]:

```
knowledgeDF.corr(method = "pearson")["UNS"].sort_values(ascending = False)
```

Out[14]:

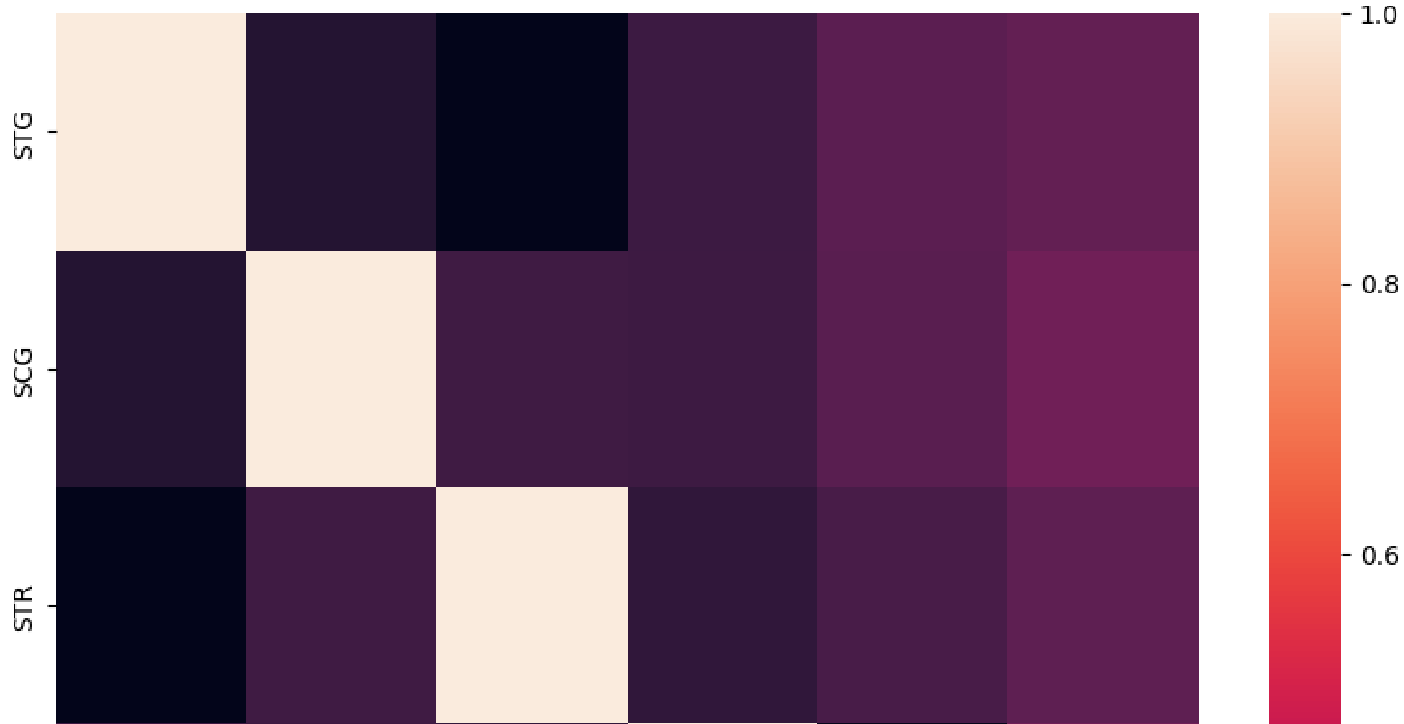
```
UNS    1.000000
PEG    0.919456
SCG    0.249095
LPR    0.247968
STG    0.217477
STR    0.203452
Name: UNS, dtype: float64
```

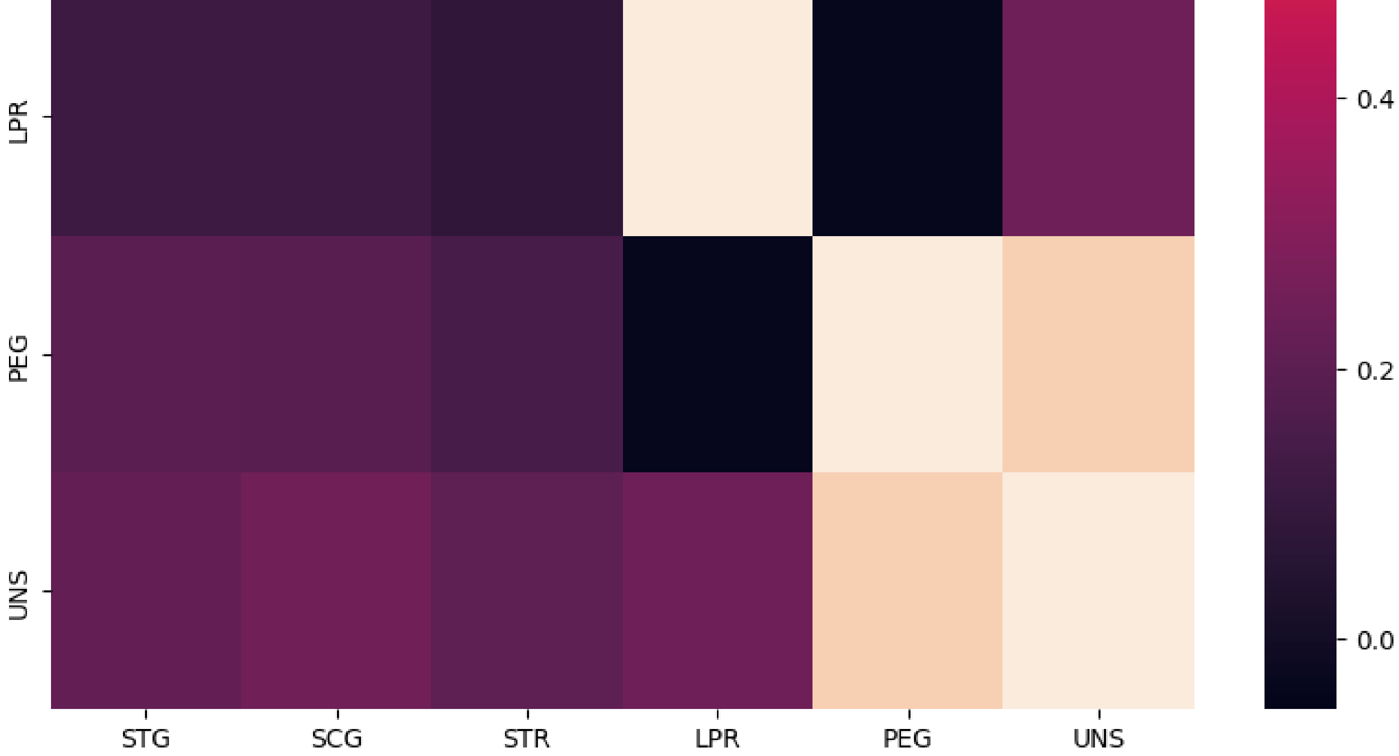
In [15]:

```
fig = plt.figure(figsize=(10,10), dpi=100)
sns.heatmap(knowledgeDF.corr())
```

Out[15]:

<Axes: >





Observation:

- We can see above that "PEG" has a very strong correlation with column "UNS". The column "PEG" corresponds to the performance of a student in an exam that is about the goal object which is about DC Electrical Machine. From here, we can infer that "PEG" has a strong correlation with the overall knowledge level of user regarding DC Electrical Machine. If the student has a high "PEG", they also have high "UNS".
- Other than that, we can observe that the predictors do not have a notable correlation with each other, meaning that we do not have to remove any predictors[2]. Data dependencies may affect the results especially if there is a high correlation between predictors.

Preparing the data and Splitting the dataset into Training, Validation and Testing sets (Classification)

- I will be splitting the dataset into 60/20/20 split, 60 for training, 20 for validation and 20 for training[3][4].
- I will be turning the target variable into categorical values using one hot encoding[5]. This is a good practice when doing multiclass classification. I will be using to_categorical from keras to easily one hot encode the target variable.

In [16]:

```
X = knowledgeDF.drop(["UNS"], axis = 1)
y = knowledgeDF["UNS"]

y_cat =  tf.keras.utils.to_categorical(y,5)

X_temp, X_test, y_temp, y_test = train_test_split(X, y_cat,
                                                  test_size =0.2,
                                                  random_state=100)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,
                                                  test_size =0.2,
                                                  random_state=110)
```

In [17]:

```
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
```

```
X_val_norm = scaler.transform(X_val)
X_test_norm = scaler.transform(X_test)
```

In [18]:

```
X_train.shape
```

Out[18]:

```
(257, 5)
```

In [19]:

```
X_test.shape
```

Out[19]:

```
(81, 5)
```

In [20]:

```
y_train.shape
```

Out[20]:

```
(257, 5)
```

Observation:

- **We can see from the results above that we have successfully split the X and Y to training and testing. We have also successfully turned the target variable to one hot encoded categorical values.**

Building the model and training the model

- **Let us build our model by using Keras Sequential model, this is a model which is built with layers stack upon each other [5]. It is very good for models with only 1 input and 1 output layer.**
- **My model will be having 1 input layer, 1 hidden layer and 1 output layer.**
- **For the parameters of my model, I used normal for the kernel initializer which is for the weights, and the activation for my input and hidden layer is both relu. For the output layer, I used softmax as the activation function as it is recommended for the multiclass classification models.**

In [21]:

```
def cl_baseline_model():

  model = tf.keras.models.Sequential([
      #input layer
      tf.keras.layers.Input((5,)),
      tf.keras.layers.Dense(5, kernel_initializer = "normal",
                           activation = "relu"),
      #hidden layer
      tf.keras.layers.Dense(3,  kernel_initializer = "normal" ,
                           activation = "relu"),
      #output layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal",
                           activation = "softmax")
  ])

  model.compile(Adam(learning_rate = 0.01),
                loss = "categorical_crossentropy",
                metrics=["accuracy"]
                )
  return model
```

```
clmodel = cl_baseline_model()

knowledgeDF_model1 = clmodel.fit(X_train_norm, y_train,
                                 validation_data = (X_val_norm, y_val) ,
                                 epochs = 100, verbose = 1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.2779 - loss: 1.5972 - val_accuracy: 0.3
385 - val_loss: 1.5566
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.2953 - loss: 1.5513 - val_accuracy: 0.3
385 - val_loss: 1.4940
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.3038 - loss: 1.4758 - val_accuracy: 0.3
846 - val_loss: 1.3826
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4140 - loss: 1.3488 - val_accuracy: 0.4
615 - val_loss: 1.2215
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5714 - loss: 1.1852 - val_accuracy: 0.5
846 - val_loss: 1.0492
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6269 - loss: 1.0052 - val_accuracy: 0.6
308 - val_loss: 0.8932
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6358 - loss: 0.8598 - val_accuracy: 0.6
000 - val_loss: 0.7901
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6378 - loss: 0.7832 - val_accuracy: 0.7
231 - val_loss: 0.7370
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7439 - loss: 0.7232 - val_accuracy: 0.8
308 - val_loss: 0.6648
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8296 - loss: 0.6285 - val_accuracy: 0.8
308 - val_loss: 0.5765
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7615 - loss: 0.6133 - val_accuracy: 0.8
462 - val_loss: 0.5143
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8543 - loss: 0.5287 - val_accuracy: 0.8
769 - val_loss: 0.4641
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8622 - loss: 0.4705 - val_accuracy: 0.8
615 - val_loss: 0.4243
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8670 - loss: 0.4366 - val_accuracy: 0.8
615 - val_loss: 0.4029
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8311 - loss: 0.4396 - val_accuracy: 0.8
615 - val_loss: 0.3708
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.8382 - loss: 0.3995 - val_accuracy: 0.8
615 - val_loss: 0.3410
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8755 - loss: 0.3552 - val_accuracy: 0.8
615 - val_loss: 0.3274
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8842 - loss: 0.3554 - val_accuracy: 0.89
23 - val_loss: 0.3302
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9460 - loss: 0.3235 - val_accuracy: 0.86
15 - val_loss: 0.2981
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8594 - loss: 0.3214 - val_accuracy: 0.8
615 - val_loss: 0.3183
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8577 - loss: 0.3213 - val_accuracy: 0.8
615 - val_loss: 0.2939
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8603 - loss: 0.2867 - val_accuracy: 0.8
615 - val_loss: 0.2777
Epoch 23/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8848 - loss: 0.2669 - val_accuracy: 0.86
15 - val_loss: 0.2635
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8642 - loss: 0.2627 - val_accuracy: 0.9
538 - val_loss: 0.2398
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9706 - loss: 0.2190 - val_accuracy: 0.9
692 - val_loss: 0.2154
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9574 - loss: 0.2074 - val_accuracy: 0.96
92 - val_loss: 0.1926
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9672 - loss: 0.1700 - val_accuracy: 0.96
92 - val_loss: 0.1711
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9740 - loss: 0.1417 - val_accuracy: 0.9
692 - val_loss: 0.1514
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9832 - loss: 0.1316 - val_accuracy: 0.9
692 - val_loss: 0.1378
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9745 - loss: 0.1338 - val_accuracy: 0.9
692 - val_loss: 0.1233
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9767 - loss: 0.1263 - val_accuracy: 0.9
692 - val_loss: 0.1358
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9846 - loss: 0.1173 - val_accuracy: 0.9
692 - val_loss: 0.1269
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9888 - loss: 0.1009 - val_accuracy: 0.9
692 - val_loss: 0.1020
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9588 - loss: 0.1099 - val_accuracy: 0.9
692 - val_loss: 0.1100
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9719 - loss: 0.1052 - val_accuracy: 0.9
692 - val_loss: 0.1170
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9675 - loss: 0.1103 - val_accuracy: 0.96
92 - val_loss: 0.1077
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9840 - loss: 0.0880 - val_accuracy: 0.96
92 - val_loss: 0.1057
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9792 - loss: 0.0777 - val_accuracy: 0.9
692 - val_loss: 0.1152
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9813 - loss: 0.0807 - val_accuracy: 0.9
692 - val_loss: 0.1230
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9679 - loss: 0.1175 - val_accuracy: 0.9
538 - val_loss: 0.1651
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9749 - loss: 0.1025 - val_accuracy: 0.9
538 - val_loss: 0.1369
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9712 - loss: 0.0879 - val_accuracy: 0.9
692 - val_loss: 0.1387
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9772 - loss: 0.0766 - val_accuracy: 0.9
692 - val_loss: 0.1195
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9771 - loss: 0.0761 - val_accuracy: 0.9
692 - val_loss: 0.1229
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.9772 - loss: 0.0819 - val_accuracy: 0.9
692 - val_loss: 0.1218
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9883 - loss: 0.0588 - val_accuracy: 0.9
692 - val_loss: 0.1191
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9702 - loss: 0.1011 - val_accuracy: 0.9
```

```
692 - val_loss: 0.1098
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9818 - loss: 0.0701 - val_accuracy: 0.9
692 - val_loss: 0.1330
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9669 - loss: 0.0954 - val_accuracy: 0.9
692 - val_loss: 0.1619
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9743 - loss: 0.0901 - val_accuracy: 0.9
692 - val_loss: 0.1234
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9604 - loss: 0.0874 - val_accuracy: 0.9
692 - val_loss: 0.1114
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9647 - loss: 0.1075 - val_accuracy: 0.9
538 - val_loss: 0.1485
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9533 - loss: 0.1286 - val_accuracy: 0.9
538 - val_loss: 0.1546
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9666 - loss: 0.1085 - val_accuracy: 0.9
692 - val_loss: 0.1630
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9904 - loss: 0.0549 - val_accuracy: 0.9
692 - val_loss: 0.1237
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9736 - loss: 0.0851 - val_accuracy: 0.9
692 - val_loss: 0.1318
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9686 - loss: 0.0748 - val_accuracy: 0.9
692 - val_loss: 0.1303
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9324 - loss: 0.2043 - val_accuracy: 0.84
62 - val_loss: 0.5750
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9184 - loss: 0.2649 - val_accuracy: 0.9
385 - val_loss: 0.2012
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9636 - loss: 0.1075 - val_accuracy: 0.9
538 - val_loss: 0.1033
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9798 - loss: 0.0659 - val_accuracy: 0.9
538 - val_loss: 0.0944
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9689 - loss: 0.0691 - val_accuracy: 0.9
692 - val_loss: 0.0995
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9838 - loss: 0.0581 - val_accuracy: 0.9
692 - val_loss: 0.1133
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9828 - loss: 0.0548 - val_accuracy: 0.9
692 - val_loss: 0.1313
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.9890 - loss: 0.0443 - val_accuracy: 0.9
692 - val_loss: 0.1343
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9782 - loss: 0.0711 - val_accuracy: 0.9
692 - val_loss: 0.1269
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.9849 - loss: 0.0547 - val_accuracy: 0.9
692 - val_loss: 0.1188
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9935 - loss: 0.0524 - val_accuracy: 0.9
692 - val_loss: 0.1231
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.9790 - loss: 0.0644 - val_accuracy: 0.9
692 - val_loss: 0.1345
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.9824 - loss: 0.0548 - val_accuracy: 0.9
692 - val_loss: 0.1288
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9832 - loss: 0.0528 - val_accuracy: 0.9
692 - val_loss: 0.1232
Epoch 72/100
```

```
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9769 - loss: 0.0537 - val_accuracy: 0.9
692 - val_loss: 0.1303
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9689 - loss: 0.0841 - val_accuracy: 0.9
692 - val_loss: 0.1420
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9735 - loss: 0.0762 - val_accuracy: 0.9
692 - val_loss: 0.1497
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9760 - loss: 0.0790 - val_accuracy: 0.9
692 - val_loss: 0.1281
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9772 - loss: 0.0835 - val_accuracy: 0.9
692 - val_loss: 0.1318
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9628 - loss: 0.0886 - val_accuracy: 0.9
077 - val_loss: 0.2618
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.9437 - loss: 0.1134 - val_accuracy: 0.9
385 - val_loss: 0.1557
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9605 - loss: 0.0798 - val_accuracy: 0.9
692 - val_loss: 0.1261
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9714 - loss: 0.0646 - val_accuracy: 0.9
538 - val_loss: 0.1327
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9799 - loss: 0.0666 - val_accuracy: 0.9
692 - val_loss: 0.1422
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9876 - loss: 0.0443 - val_accuracy: 0.9
692 - val_loss: 0.1578
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9275 - loss: 0.2027 - val_accuracy: 0.9
538 - val_loss: 0.3003
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9098 - loss: 0.2527 - val_accuracy: 0.9
538 - val_loss: 0.2595
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9375 - loss: 0.1370 - val_accuracy: 0.9
692 - val_loss: 0.2223
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9877 - loss: 0.0765 - val_accuracy: 0.9
692 - val_loss: 0.1884
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9742 - loss: 0.0981 - val_accuracy: 0.9
692 - val_loss: 0.1604
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9674 - loss: 0.0820 - val_accuracy: 0.9
692 - val_loss: 0.1563
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9722 - loss: 0.0725 - val_accuracy: 0.9
538 - val_loss: 0.1694
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9792 - loss: 0.0568 - val_accuracy: 0.96
92 - val_loss: 0.1607
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9894 - loss: 0.0431 - val_accuracy: 0.9
692 - val_loss: 0.1564
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9790 - loss: 0.0522 - val_accuracy: 0.9
692 - val_loss: 0.1438
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9827 - loss: 0.0473 - val_accuracy: 0.9
692 - val_loss: 0.1435
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9803 - loss: 0.0458 - val_accuracy: 0.9
692 - val_loss: 0.1322
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9794 - loss: 0.0486 - val_accuracy: 0.9
692 - val_loss: 0.1412
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9876 - loss: 0.0405 - val_accuracy: 0.9
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9876 - loss: 0.0405 - val_accuracy: 0.9
692 - val_loss: 0.1405
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9847 - loss: 0.0449 - val_accuracy: 0.9
692 - val_loss: 0.1410
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9812 - loss: 0.0628 - val_accuracy: 0.9
692 - val_loss: 0.1424
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9833 - loss: 0.0432 - val_accuracy: 0.96
92 - val_loss: 0.1404
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9794 - loss: 0.0494 - val_accuracy: 0.96
92 - val_loss: 0.1350
```

In [22]:

```
clmodel.evaluate(X_test_norm, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9620 - loss: 0.1167
```

Out[22]:

```
[0.10060442239046097, 0.9629629850387573]
```

Observation:

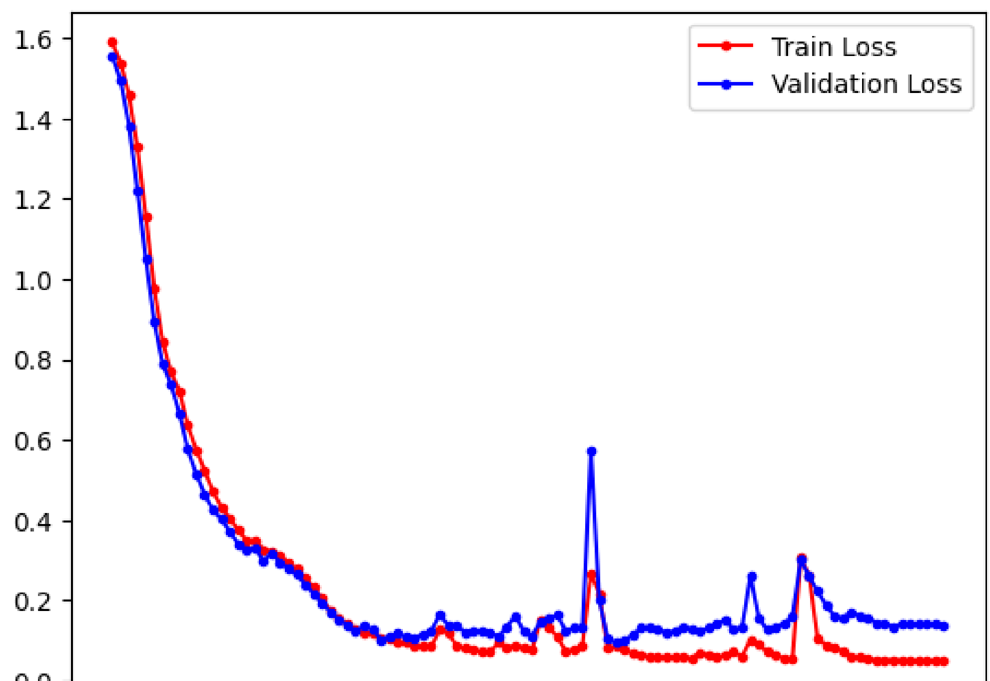
- **We can see from the results above that the model is well-fitted to the dataset. The final training accuracy is (0.9794) with a loss of (0.0494) and the final validation accuracy is (0.9692) with a val loss of (0.1350). This is a pretty good result for this dataset by just doing some simple standardization.**
- **We can also observe the .evaluate() method which evaluates our model by generating loss and accuracy based on the testing dataset. We can see here that the result is very good and this shows that the model is well-fitted and not overfitted. The testing accuracy is (0.9629) and the testing loss is (0.107).**
- **Overall, this model is well-fitted and it gives high accuracy in prediction and low loss.**

In [23]:

```
fig, ax = plt.subplots()
ax.plot(knowledgeDF_model1.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(knowledgeDF_model1.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[23]:

```
<matplotlib.legend.Legend at 0x7eaa36e872e0>
```

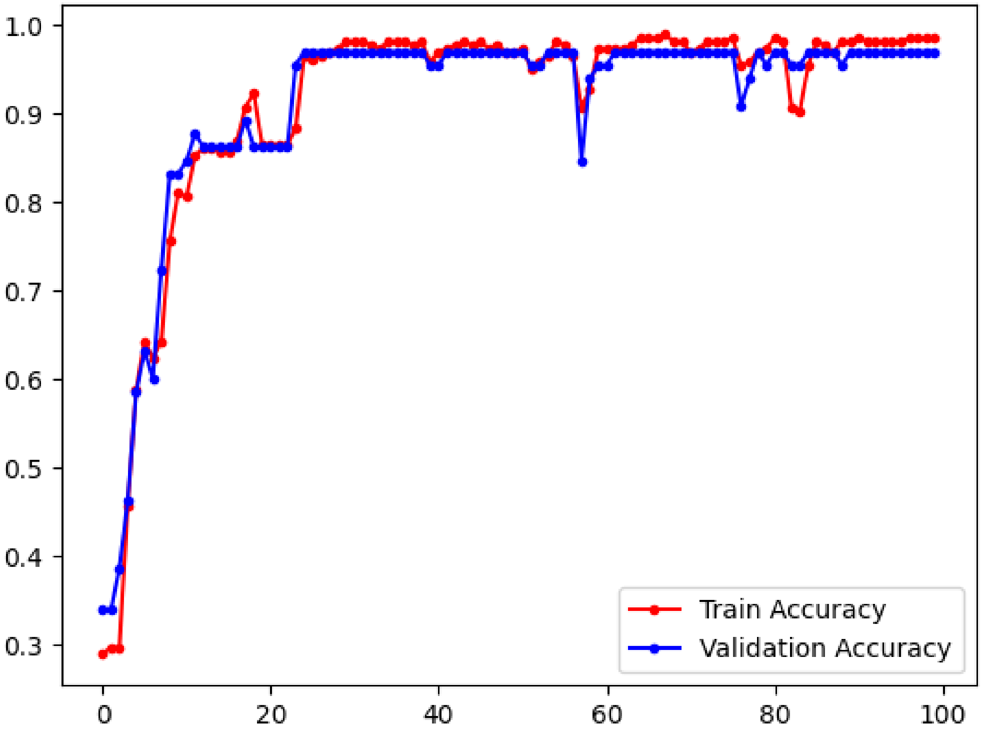


In [24]:

```
fig, ax = plt.subplots()
ax.plot(knowledgeDF_model1.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(knowledgeDF_model1.history["val_accuracy"],'b', marker='.', label="Validation Accu
racy")
ax.legend()
```

Out[24]:

```
<matplotlib.legend.Legend at 0x7eaa235e0ca0>
```



In [34]:

```
estimators = [('standardize', StandardScaler()),
              ('mlp', KerasClassifier(model = cl_baseline_model, epochs=100,
                                     verbose = 0))]

pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)

results0 = cross_val_score(pipeline, X = X, y = y_cat, cv=kfold, scoring='accuracy')
print("kFold: %.4f (%.4f) Accuracy" % (results0.mean(), results0.std()))
```

```
kFold: 0.9556 (0.0520) Accuracy
```

Observation:

- **To further validate the accuracy of this model, I utilized kFold cross validation to look at the mean of the results and the standard deviation.**
- **We can see above that the mean accuracy for the kFold cross validation is (0.9556) and std of (0.0520). This is a good result considering that we used 10 folds and we get the mean for the 10 results of the kFold.**
- **This confirms that our model is well-fitted with the dataset although there is a slight discrepancy with the training accuracy and kfold accuracy.**

Saving the Model (Evidences)

Saving the Model in HDF5

- **The HDF5 format is a format used for storing large amounts of data[7]. We can use HDF5 format by importing the h5py library to our coding project.**
- **We will be saving our entire model in the HDF5 format so that we can use it later and load it when we need it.**

In [27]:

```
!pip install h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (3.9.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (fr
om h5py) (1.25.2)
```

In [26]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [48]:

```
scores = clmodel.evaluate(X_test_norm, y_test, verbose=0)
print("\n%s: %.2f%%" % (clmodel.metrics_names[1], scores[1]*100))
clmodel.save("model.h5")
print("Saved model to disk")

print("\nLoaded model from disk")
clloaded_model = load_model('model.h5')
print("%s: %.2f%%" % (clloaded_model.metrics_names[1], scores[1]*100))
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.
save_model(model)`. This file format is considered legacy. We recommend using instead the n
ative Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `mod
el.compile_metrics` will be empty until you train or evaluate the model.
```

```
compile_metrics: 96.30%
Saved model to disk

Loaded model from disk
compile_metrics: 96.30%
```

Observation:

- **As you can observe above, I first stored the results of the .evaluate() to a variable and then we also displayed the accuracy by accessing that variable. Then after that, we printed that accuracy to display the accuracy of the model before saving it. Then we saved the model using .save() and using the extension h5 to save the model. This essentially saves the whole model, the architecture and the weights. To load the model, we will use load_model() to load the "model.h5", and then print the score of the loaded models.**
- **Overall, this HDF5 format is concise in saving the model and it is simple. It saves the model in 1 file and then we can easily load it using built-in methods.**

Saving Model in JSON

- **Saving your model in JSON differs in HDF5 because we need to separately save the model architecture and the model weights[8].**

In [53]:

```
scores = clmodel.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (clmodel.metrics_names[1], scores[1]*100))


clmodel_json = clmodel.to_json()
with open("model.json", "w") as json_file:
    json_file.write(clmodel_json)
clmodel.save_weights("/content/drive/MyDrive/Colab Notebooks/model.weights.h5")
print("Saved model to disk")


json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("/content/drive/MyDrive/Colab Notebooks/model.weights.h5")
print("\nLoaded model from disk")

loaded_model.compile(Adam(learning_rate = 0.01),loss='categorical_crossentropy', metrics=[
'accuracy'])
score = loaded_model.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
compile_metrics: 96.30%
Saved model to disk

Loaded model from disk
compile_metrics: 96.30%
```

Observation:

- **We also stored the results of the .evaluate() to a variable then printed the accuracy score before saving. This is to prove that we can use to compare the saved and the loaded model. We first saved the model architecture by using .to_json() and then we saved model weights by using .save_weights. To open this, we can use open() and then the name of "model.json". We can read this file using .read() and then load the model using model_from_json. We can add the weights to the loaded model using .load_weights.**
- **Overall, saving in JSON seems complicated but this gives us more control with our model and flexibility by changing the weights at will unlike the HDF5.**

Saving Model in YAML

- **Here in YAML, it is essentially the same as JSON where you need to save the model architecture and the model weights separately[8]. YAML is mostly used in model configurations.**

In [81]:

```
!pip install PyYAML
```

```
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (6.0.1)
```

In [86]:

```
scores = clmodel.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (clmodel.metrics_names[1], scores[1]*100))
clmodel_yaml = clmodel.to_json()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(clmodel_yaml)
clmodel.save_weights("/content/drive/MyDrive/Colab Notebooks/model_yaml.weights.h5")
print("Saved model to disk")


yaml_file = open('model.yaml', 'r')
```

```
loaded_model_yaml = yaml_file.read()
yaml_file.close()
loaded_model = model_from_json(loaded_model_yaml)
loaded_model.load_weights("/content/drive/MyDrive/Colab Notebooks/model_yaml.weights.h5")
print("\nLoaded model from disk")

loaded_model.compile(Adam(learning_rate = 0.01),loss='categorical_crossentropy', metrics=[
'accuracy'])
score = loaded_model.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
compile_metrics: 96.30%
Saved model to disk

Loaded model from disk
compile_metrics: 96.30%
```

Observation:

- **We can see from above that the output is essentially the same with the JSON and the accuracy for the saved and the load model is the same. The difference is that we used yaml_file instead of json_file.**

Checkpoint Neural Network Improvements

- **Here, we are saving selectively by using the validation accuracy as a threshold before saving. We are monitoring the validation accuracy and we are only saving the model when the validation accuracy increased.**
- **The values saved here are epochs, val accuracy and the weights.**

In [59]:

```
checkpoint_model = cl_baseline_model()

filepath="weights-improvement-{epoch:02d}-{val_accuracy:.2f}.hdf5.keras"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='max')
callbacks_list = [checkpoint]

# Fit the model
checkpoint_model.fit(X_train_norm, y_train, validation_data = (X_val_norm, y_val),
                     epochs=100, callbacks=callbacks_list, verbose=0)
```

```
Epoch 1: val_accuracy improved from -inf to 0.33846, saving model to weights-improvement-01
-0.34.hdf5.keras

Epoch 2: val_accuracy did not improve from 0.33846

Epoch 3: val_accuracy did not improve from 0.33846

Epoch 4: val_accuracy did not improve from 0.33846

Epoch 5: val_accuracy improved from 0.33846 to 0.53846, saving model to weights-improvement
-05-0.54.hdf5.keras

Epoch 6: val_accuracy improved from 0.53846 to 0.63077, saving model to weights-improvement
-06-0.63.hdf5.keras

Epoch 7: val_accuracy did not improve from 0.63077

Epoch 8: val_accuracy did not improve from 0.63077

Epoch 9: val_accuracy improved from 0.63077 to 0.73846, saving model to weights-improvement
-09-0.74.hdf5.keras

Epoch 10: val_accuracy improved from 0.73846 to 0.80000, saving model to weights-improvemen
t-10-0.80.hdf5.keras
```

```
Epoch 11: val_accuracy improved from 0.80000 to 0.83077, saving model to weights-improvemen
t-11-0.83.hdf5.keras

Epoch 12: val_accuracy improved from 0.83077 to 0.87692, saving model to weights-improvemen
t-12-0.88.hdf5.keras

Epoch 13: val_accuracy did not improve from 0.87692

Epoch 14: val_accuracy did not improve from 0.87692

Epoch 15: val_accuracy did not improve from 0.87692

Epoch 16: val_accuracy did not improve from 0.87692

Epoch 17: val_accuracy did not improve from 0.87692

Epoch 18: val_accuracy did not improve from 0.87692

Epoch 19: val_accuracy did not improve from 0.87692

Epoch 20: val_accuracy improved from 0.87692 to 0.89231, saving model to weights-improvemen
t-20-0.89.hdf5.keras

Epoch 21: val_accuracy did not improve from 0.89231

Epoch 22: val_accuracy improved from 0.89231 to 0.96923, saving model to weights-improvemen
t-22-0.97.hdf5.keras

Epoch 23: val_accuracy did not improve from 0.96923

Epoch 24: val_accuracy did not improve from 0.96923

Epoch 25: val_accuracy did not improve from 0.96923

Epoch 26: val_accuracy did not improve from 0.96923

Epoch 27: val_accuracy did not improve from 0.96923

Epoch 28: val_accuracy did not improve from 0.96923

Epoch 29: val_accuracy did not improve from 0.96923

Epoch 30: val_accuracy did not improve from 0.96923

Epoch 31: val_accuracy did not improve from 0.96923

Epoch 32: val_accuracy did not improve from 0.96923

Epoch 33: val_accuracy did not improve from 0.96923

Epoch 34: val_accuracy did not improve from 0.96923

Epoch 35: val_accuracy did not improve from 0.96923

Epoch 36: val_accuracy did not improve from 0.96923

Epoch 37: val_accuracy did not improve from 0.96923

Epoch 38: val_accuracy did not improve from 0.96923

Epoch 39: val_accuracy did not improve from 0.96923

Epoch 40: val_accuracy did not improve from 0.96923

Epoch 41: val_accuracy did not improve from 0.96923

Epoch 42: val_accuracy did not improve from 0.96923

Epoch 43: val_accuracy did not improve from 0.96923

Epoch 44: val_accuracy did not improve from 0.96923
```

```
Epoch 45: val_accuracy did not improve from 0.96923

Epoch 46: val_accuracy did not improve from 0.96923

Epoch 47: val_accuracy did not improve from 0.96923

Epoch 48: val_accuracy did not improve from 0.96923

Epoch 49: val_accuracy did not improve from 0.96923

Epoch 50: val_accuracy did not improve from 0.96923

Epoch 51: val_accuracy did not improve from 0.96923

Epoch 52: val_accuracy did not improve from 0.96923

Epoch 53: val_accuracy did not improve from 0.96923

Epoch 54: val_accuracy did not improve from 0.96923

Epoch 55: val_accuracy did not improve from 0.96923

Epoch 56: val_accuracy did not improve from 0.96923

Epoch 57: val_accuracy did not improve from 0.96923

Epoch 58: val_accuracy did not improve from 0.96923

Epoch 59: val_accuracy did not improve from 0.96923

Epoch 60: val_accuracy did not improve from 0.96923

Epoch 61: val_accuracy did not improve from 0.96923

Epoch 62: val_accuracy did not improve from 0.96923

Epoch 63: val_accuracy did not improve from 0.96923

Epoch 64: val_accuracy did not improve from 0.96923

Epoch 65: val_accuracy did not improve from 0.96923

Epoch 66: val_accuracy did not improve from 0.96923

Epoch 67: val_accuracy did not improve from 0.96923

Epoch 68: val_accuracy did not improve from 0.96923

Epoch 69: val_accuracy did not improve from 0.96923

Epoch 70: val_accuracy did not improve from 0.96923

Epoch 71: val_accuracy did not improve from 0.96923

Epoch 72: val_accuracy did not improve from 0.96923

Epoch 73: val_accuracy did not improve from 0.96923

Epoch 74: val_accuracy did not improve from 0.96923

Epoch 75: val_accuracy did not improve from 0.96923

Epoch 76: val_accuracy did not improve from 0.96923

Epoch 77: val_accuracy did not improve from 0.96923

Epoch 78: val_accuracy did not improve from 0.96923

Epoch 79: val_accuracy did not improve from 0.96923

Epoch 80: val_accuracy did not improve from 0.96923

Epoch 81: val_accuracy did not improve from 0.96923
```

```
Epoch 82: val_accuracy did not improve from 0.96923

Epoch 83: val_accuracy did not improve from 0.96923

Epoch 84: val_accuracy did not improve from 0.96923

Epoch 85: val_accuracy did not improve from 0.96923

Epoch 86: val_accuracy did not improve from 0.96923

Epoch 87: val_accuracy did not improve from 0.96923

Epoch 88: val_accuracy did not improve from 0.96923

Epoch 89: val_accuracy did not improve from 0.96923

Epoch 90: val_accuracy did not improve from 0.96923

Epoch 91: val_accuracy did not improve from 0.96923

Epoch 92: val_accuracy did not improve from 0.96923

Epoch 93: val_accuracy did not improve from 0.96923

Epoch 94: val_accuracy did not improve from 0.96923

Epoch 95: val_accuracy did not improve from 0.96923

Epoch 96: val_accuracy did not improve from 0.96923

Epoch 97: val_accuracy did not improve from 0.96923

Epoch 98: val_accuracy did not improve from 0.96923

Epoch 99: val_accuracy did not improve from 0.96923

Epoch 100: val_accuracy did not improve from 0.96923
```

Out[59]:

```
<keras.src.callbacks.history.History at 0x7eaa1a185a50>
```

Observation:

- **We can observe from the results above that there were 9 checkpoints that happened in this model training. The checkpoints happened in 1st epoch, 5th epoch, 6th epoch, 9th epoch, 10th epoch, 11th epoch, 20th and 22nd epoch. From the initial 0.3386, it became 0.96923 in the 22nd epoch. The beauty of this feature is that the model will not regress to a worst accuracy and you will also be able to save space by only saving the high validation accuracy.**

Checkpoint Best Neural Network Model only

- **The difference between this and the code above is that this code only saves the best weights and not the val accuracy and the epochs.**

In [88]:

```
checkpoint_model2 = cl_baseline_model()

filepath="weights.best.hdf5.keras"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

```
checkpoint_model2.fit(X_train_norm, y_train,
                      validation_data = (X_val_norm, y_val),
                     epochs=100, callbacks=callbacks_list, verbose=0)
```

```
Epoch 1: val_accuracy improved from -inf to 0.33846, saving model to weights.best.hdf5.kera
s

Epoch 2: val_accuracy improved from 0.33846 to 0.44615, saving model to weights.best.hdf5.k
eras

Epoch 3: val_accuracy improved from 0.44615 to 0.50769, saving model to weights.best.hdf5.k
eras

Epoch 4: val_accuracy did not improve from 0.50769

Epoch 5: val_accuracy did not improve from 0.50769

Epoch 6: val_accuracy did not improve from 0.50769

Epoch 7: val_accuracy did not improve from 0.50769

Epoch 8: val_accuracy improved from 0.50769 to 0.56923, saving model to weights.best.hdf5.k
eras

Epoch 9: val_accuracy improved from 0.56923 to 0.64615, saving model to weights.best.hdf5.k
eras

Epoch 10: val_accuracy did not improve from 0.64615

Epoch 11: val_accuracy did not improve from 0.64615

Epoch 12: val_accuracy did not improve from 0.64615

Epoch 13: val_accuracy improved from 0.64615 to 0.83077, saving model to weights.best.hdf5.
keras

Epoch 14: val_accuracy did not improve from 0.83077

Epoch 15: val_accuracy did not improve from 0.83077

Epoch 16: val_accuracy did not improve from 0.83077

Epoch 17: val_accuracy improved from 0.83077 to 0.86154, saving model to weights.best.hdf5.
keras

Epoch 18: val_accuracy did not improve from 0.86154

Epoch 19: val_accuracy did not improve from 0.86154

Epoch 20: val_accuracy did not improve from 0.86154

Epoch 21: val_accuracy did not improve from 0.86154

Epoch 22: val_accuracy did not improve from 0.86154

Epoch 23: val_accuracy did not improve from 0.86154

Epoch 24: val_accuracy did not improve from 0.86154

Epoch 25: val_accuracy did not improve from 0.86154

Epoch 26: val_accuracy did not improve from 0.86154

Epoch 27: val_accuracy did not improve from 0.86154

Epoch 28: val_accuracy did not improve from 0.86154

Epoch 29: val_accuracy did not improve from 0.86154

Epoch 30: val_accuracy did not improve from 0.86154

Epoch 31: val_accuracy did not improve from 0.86154
```

```
Epoch 32: val_accuracy did not improve from 0.86154

Epoch 33: val_accuracy did not improve from 0.86154

Epoch 34: val_accuracy did not improve from 0.86154

Epoch 35: val_accuracy did not improve from 0.86154

Epoch 36: val_accuracy did not improve from 0.86154

Epoch 37: val_accuracy did not improve from 0.86154

Epoch 38: val_accuracy did not improve from 0.86154

Epoch 39: val_accuracy did not improve from 0.86154

Epoch 40: val_accuracy did not improve from 0.86154

Epoch 41: val_accuracy did not improve from 0.86154

Epoch 42: val_accuracy did not improve from 0.86154

Epoch 43: val_accuracy did not improve from 0.86154

Epoch 44: val_accuracy did not improve from 0.86154

Epoch 45: val_accuracy did not improve from 0.86154

Epoch 46: val_accuracy did not improve from 0.86154

Epoch 47: val_accuracy did not improve from 0.86154

Epoch 48: val_accuracy did not improve from 0.86154

Epoch 49: val_accuracy did not improve from 0.86154

Epoch 50: val_accuracy did not improve from 0.86154

Epoch 51: val_accuracy did not improve from 0.86154

Epoch 52: val_accuracy did not improve from 0.86154

Epoch 53: val_accuracy did not improve from 0.86154

Epoch 54: val_accuracy did not improve from 0.86154

Epoch 55: val_accuracy did not improve from 0.86154

Epoch 56: val_accuracy did not improve from 0.86154

Epoch 57: val_accuracy did not improve from 0.86154

Epoch 58: val_accuracy did not improve from 0.86154

Epoch 59: val_accuracy did not improve from 0.86154

Epoch 60: val_accuracy did not improve from 0.86154

Epoch 61: val_accuracy did not improve from 0.86154

Epoch 62: val_accuracy did not improve from 0.86154

Epoch 63: val_accuracy did not improve from 0.86154

Epoch 64: val_accuracy did not improve from 0.86154

Epoch 65: val_accuracy did not improve from 0.86154

Epoch 66: val_accuracy did not improve from 0.86154

Epoch 67: val_accuracy did not improve from 0.86154
```

```
Epoch 68: val_accuracy did not improve from 0.86154

Epoch 69: val_accuracy did not improve from 0.86154

Epoch 70: val_accuracy did not improve from 0.86154

Epoch 71: val_accuracy did not improve from 0.86154

Epoch 72: val_accuracy did not improve from 0.86154

Epoch 73: val_accuracy did not improve from 0.86154

Epoch 74: val_accuracy did not improve from 0.86154

Epoch 75: val_accuracy did not improve from 0.86154

Epoch 76: val_accuracy did not improve from 0.86154

Epoch 77: val_accuracy did not improve from 0.86154

Epoch 78: val_accuracy did not improve from 0.86154

Epoch 79: val_accuracy did not improve from 0.86154

Epoch 80: val_accuracy did not improve from 0.86154

Epoch 81: val_accuracy did not improve from 0.86154

Epoch 82: val_accuracy did not improve from 0.86154

Epoch 83: val_accuracy did not improve from 0.86154

Epoch 84: val_accuracy did not improve from 0.86154

Epoch 85: val_accuracy did not improve from 0.86154

Epoch 86: val_accuracy did not improve from 0.86154

Epoch 87: val_accuracy did not improve from 0.86154

Epoch 88: val_accuracy did not improve from 0.86154

Epoch 89: val_accuracy did not improve from 0.86154

Epoch 90: val_accuracy did not improve from 0.86154

Epoch 91: val_accuracy did not improve from 0.86154

Epoch 92: val_accuracy did not improve from 0.86154

Epoch 93: val_accuracy did not improve from 0.86154

Epoch 94: val_accuracy did not improve from 0.86154

Epoch 95: val_accuracy did not improve from 0.86154

Epoch 96: val_accuracy did not improve from 0.86154

Epoch 97: val_accuracy did not improve from 0.86154

Epoch 98: val_accuracy did not improve from 0.86154

Epoch 99: val_accuracy did not improve from 0.86154

Epoch 100: val_accuracy did not improve from 0.86154
```

Out[88]:

```
<keras.src.callbacks.history.History at 0x7eaa1bd16fe0>
```

Observation:

- In this model training, there are 6 checkpoints that happened throughout 100 epochs. In particular, the checkpoints happened in epochs 1,4,7,8,11,12 and 76.

Load a saved Neural Network model

- We can load any model weights that we want to load and we can get consistent accuracies with it.
- We will load the weights and then use it as the weights of our baseline model for the first code block. For the next code blocks, I will be using JSON and YAML.

In [97]:

```
clload = cl_baseline_model()
clload.load_weights("/content/weights.best.hdf5.keras")

scores = clload.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (clload.metrics_names[1], scores[1]*100))
```

```
compile_metrics: 77.78%
```

In [100]:

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("/content/weights-improvement-20-0.89.hdf5.keras")

scores = loaded_model.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

```
compile_metrics: 95.06%
```

In [101]:

```
yaml_file = open('model.yaml', 'r')
loaded_model_yaml = yaml_file.read()
yaml_file.close()
loaded_model = model_from_json(loaded_model_yaml)
loaded_model.load_weights("/content/weights-improvement-22-0.97.hdf5.keras")

scores = loaded_model.evaluate(X_test_norm, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], scores[1]*100))
```

```
compile_metrics: 92.59%
```

Observation:

- We can observe above the different ways of loading weights and model architecture. We can also observe the different file formats that was used to load the weights and the model above. All the loaded model and weights are the ones saved in the earlier codes.

Visualizing the Model Training in Keras

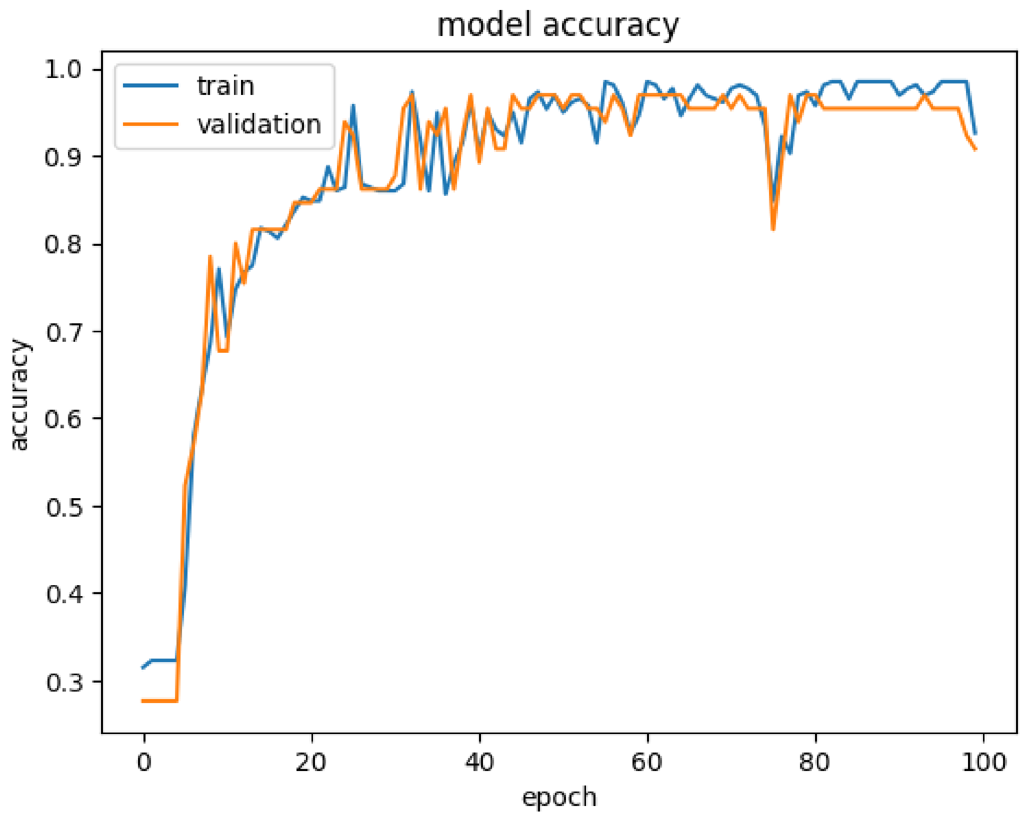
- When we are training a model, we can store it in a variable so that we can access the history keys of the particular model that we trained. We can see the differnet history keys that we can access in the printed values below.
- History keys saves the values of the loss and accuracy for every epochs. This can act as our monitor to observe the behavior during the model training.

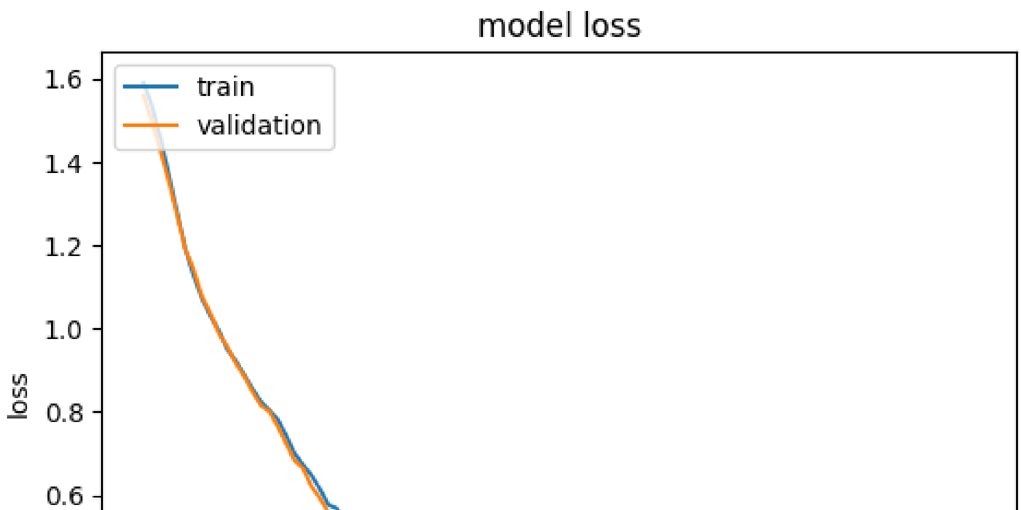
In [126]:

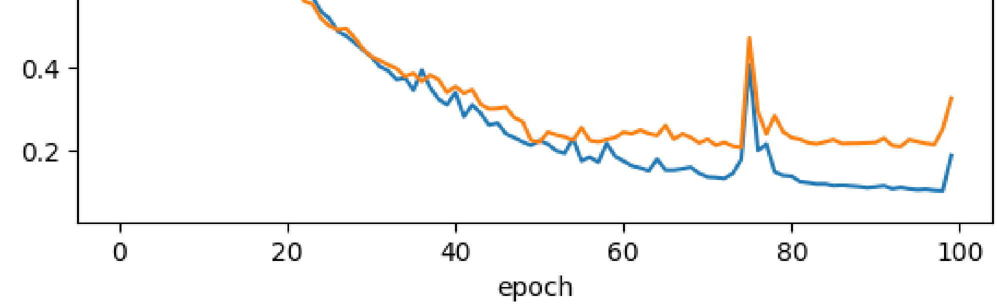
```
def visualize_history(model):
    history = model
    # list all data in history
    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

visualize_history(knowledgeDF_model1)
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







Observation:

- **We can see above the graph for the baseline model accuracy and baseline model loss. The graph above compares the training and validation accuracy and loss from epoch 0 to epoch 100.**
- **We can observe above that there are some fluctuations with the values of the accuracy but the overall trend is that it is increasing for both training and validation. It is also not overfitted since the gap between the 2 lines are not that far. For the loss, there are also fluctuations but lesser than the accuracy, the overall trend of loss is that it is decreasing as the epochs increases.**
- **Overall, this visualization is useful so that we can see where are the aspects that we can improve with our model. We can also see that optimum amount of epochs that can achieve the results that we want, which can either increase or decrease the time for training.**

Dropout regularization in Neural Network

Dropout layer in Input Layer

- **Dropout regularization is a regularization technique where you input a probability and then this will randomly drop nodes which can reduce overfitting and co-adapting of the model [9]. I also used a value of 0.3 for the dropout as it is stated in the paper that it is within the acceptable value.**

In [67]:

```
def cl_dropout_model():

  model = tf.keras.models.Sequential([
      #input layer
      tf.keras.layers.Dropout(0.3, input_shape=(5,)),
      #hidden layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal" ,
                             activation = "relu"),
      tf.keras.layers.Dense(3,  kernel_initializer = "normal" ,
                             activation = "relu"),
      #output layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal",
                             activation = "softmax")
  ])

  model.compile(Adam(learning_rate = 0.01),
               loss = "categorical_crossentropy",
               metrics=["accuracy"]
               )
  return model

clmodel2 = cl_dropout_model()

knowledgeDF_model3 = clmodel2.fit(X_train_norm, y_train,
                                  validation_data = (X_val_norm, y_val) ,
                                  epochs = 100, verbose = 1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.2621 - loss: 1.5995 - val_accuracy: 0.2
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.2621 - loss: 1.5995 - val_accuracy: 0.2
769 - val_loss: 1.5632
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2930 - loss: 1.5535 - val_accuracy: 0.2
769 - val_loss: 1.5199
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3163 - loss: 1.5041 - val_accuracy: 0.2
769 - val_loss: 1.4783
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.3400 - loss: 1.4585 - val_accuracy: 0.2
769 - val_loss: 1.4372
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3281 - loss: 1.4135 - val_accuracy: 0.4
769 - val_loss: 1.3773
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3761 - loss: 1.3857 - val_accuracy: 0.3
385 - val_loss: 1.3163
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.3013 - loss: 1.3217 - val_accuracy: 0.3
077 - val_loss: 1.2443
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3987 - loss: 1.2661 - val_accuracy: 0.47
69 - val_loss: 1.1930
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4623 - loss: 1.2243 - val_accuracy: 0.53
85 - val_loss: 1.1372
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5189 - loss: 1.2189 - val_accuracy: 0.55
38 - val_loss: 1.0872
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5821 - loss: 1.1461 - val_accuracy: 0.55
38 - val_loss: 1.0354
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5468 - loss: 1.1263 - val_accuracy: 0.64
62 - val_loss: 0.9953
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5229 - loss: 1.0908 - val_accuracy: 0.73
85 - val_loss: 0.9614
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5804 - loss: 1.0954 - val_accuracy: 0.64
62 - val_loss: 0.9301
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5478 - loss: 1.0788 - val_accuracy: 0.73
85 - val_loss: 0.9135
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5293 - loss: 1.0454 - val_accuracy: 0.69
23 - val_loss: 0.8965
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5970 - loss: 1.0243 - val_accuracy: 0.81
54 - val_loss: 0.8864
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5639 - loss: 1.0504 - val_accuracy: 0.83
08 - val_loss: 0.8776
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6144 - loss: 1.0105 - val_accuracy: 0.84
62 - val_loss: 0.8649
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6399 - loss: 0.9645 - val_accuracy: 0.81
54 - val_loss: 0.8313
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6024 - loss: 0.9566 - val_accuracy: 0.69
23 - val_loss: 0.8129
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5365 - loss: 1.0402 - val_accuracy: 0.80
00 - val_loss: 0.8095
Epoch 23/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5565 - loss: 0.9690 - val_accuracy: 0.83
08 - val_loss: 0.8026
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6157 - loss: 0.9444 - val_accuracy: 0.83
08 - val_loss: 0.7876
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6417 - loss: 0.8954 - val_accuracy: 0.83
08 - val_loss: 0.7762
```

```
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5957 - loss: 0.9109 - val_accuracy: 0.73
85 - val_loss: 0.7602
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5209 - loss: 0.9729 - val_accuracy: 0.80
00 - val_loss: 0.7738
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6178 - loss: 0.9561 - val_accuracy: 0.64
62 - val_loss: 0.8040
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6593 - loss: 0.9577 - val_accuracy: 0.83
08 - val_loss: 0.7459
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5672 - loss: 0.9603 - val_accuracy: 0.84
62 - val_loss: 0.7329
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5913 - loss: 0.8785 - val_accuracy: 0.83
08 - val_loss: 0.7328
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5587 - loss: 0.9103 - val_accuracy: 0.78
46 - val_loss: 0.7674
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6271 - loss: 0.8951 - val_accuracy: 0.78
46 - val_loss: 0.7636
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6156 - loss: 0.9633 - val_accuracy: 0.67
69 - val_loss: 0.7882
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6520 - loss: 0.9396 - val_accuracy: 0.80
00 - val_loss: 0.7273
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6241 - loss: 0.8920 - val_accuracy: 0.67
69 - val_loss: 0.7172
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5370 - loss: 0.9228 - val_accuracy: 0.78
46 - val_loss: 0.7522
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5801 - loss: 0.9115 - val_accuracy: 0.76
92 - val_loss: 0.7621
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5563 - loss: 0.9405 - val_accuracy: 0.78
46 - val_loss: 0.7413
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5252 - loss: 0.9696 - val_accuracy: 0.78
46 - val_loss: 0.7343
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5897 - loss: 0.9007 - val_accuracy: 0.78
46 - val_loss: 0.7238
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5775 - loss: 0.9250 - val_accuracy: 0.78
46 - val_loss: 0.7184
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5424 - loss: 0.9026 - val_accuracy: 0.80
00 - val_loss: 0.7181
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5865 - loss: 0.9073 - val_accuracy: 0.80
00 - val_loss: 0.7124
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6520 - loss: 0.8369 - val_accuracy: 0.76
92 - val_loss: 0.7181
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6568 - loss: 0.8586 - val_accuracy: 0.78
46 - val_loss: 0.7145
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6366 - loss: 0.8827 - val_accuracy: 0.78
46 - val_loss: 0.7132
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6317 - loss: 0.8719 - val_accuracy: 0.78
46 - val_loss: 0.7125
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6138 - loss: 0.8888 - val_accuracy: 0.80
00 - val_loss: 0.6820
Epoch 50/100
```

```
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5434 - loss: 0.9361 - val_accuracy: 0.80
00 - val_loss: 0.6795
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5511 - loss: 0.8741 - val_accuracy: 0.78
46 - val_loss: 0.6758
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5787 - loss: 0.8951 - val_accuracy: 0.78
46 - val_loss: 0.6729
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5479 - loss: 0.9509 - val_accuracy: 0.8
000 - val_loss: 0.6679
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6668 - loss: 0.8238 - val_accuracy: 0.72
31 - val_loss: 0.7086
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6511 - loss: 0.9087 - val_accuracy: 0.80
00 - val_loss: 0.6869
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6748 - loss: 0.8239 - val_accuracy: 0.8
154 - val_loss: 0.6615
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6396 - loss: 0.8396 - val_accuracy: 0.83
08 - val_loss: 0.6445
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5842 - loss: 0.8642 - val_accuracy: 0.86
15 - val_loss: 0.6418
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6279 - loss: 0.8681 - val_accuracy: 0.84
62 - val_loss: 0.6546
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.6271 - loss: 0.8746 - val_accuracy: 0.8
462 - val_loss: 0.6340
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.6740 - loss: 0.8504 - val_accuracy: 0.8
615 - val_loss: 0.6322
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.5994 - loss: 0.9187 - val_accuracy: 0.8
462 - val_loss: 0.6364
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.6307 - loss: 0.8667 - val_accuracy: 0.8
462 - val_loss: 0.6396
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6531 - loss: 0.8247 - val_accuracy: 0.8
615 - val_loss: 0.6365
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.6075 - loss: 0.8591 - val_accuracy: 0.8
615 - val_loss: 0.6403
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.6577 - loss: 0.8142 - val_accuracy: 0.8
308 - val_loss: 0.6426
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5657 - loss: 0.9121 - val_accuracy: 0.7
385 - val_loss: 0.6900
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6613 - loss: 0.8588 - val_accuracy: 0.8
154 - val_loss: 0.6806
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6506 - loss: 0.8426 - val_accuracy: 0.8
462 - val_loss: 0.6691
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6123 - loss: 0.8309 - val_accuracy: 0.80
00 - val_loss: 0.6518
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5731 - loss: 0.9365 - val_accuracy: 0.8
308 - val_loss: 0.6481
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6665 - loss: 0.8543 - val_accuracy: 0.8
308 - val_loss: 0.6479
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6331 - loss: 0.8387 - val_accuracy: 0.8
000 - val_loss: 0.6452
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6008 - loss: 0.8781 - val_accuracy: 0.8
```

```
308 - val_loss: 0.6322
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6044 - loss: 0.8789 - val_accuracy: 0.84
62 - val_loss: 0.6255
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6418 - loss: 0.8891 - val_accuracy: 0.80
00 - val_loss: 0.6468
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6770 - loss: 0.8278 - val_accuracy: 0.76
92 - val_loss: 0.6552
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6589 - loss: 0.8852 - val_accuracy: 0.8
154 - val_loss: 0.6396
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6640 - loss: 0.8318 - val_accuracy: 0.81
54 - val_loss: 0.6233
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6585 - loss: 0.7991 - val_accuracy: 0.8
000 - val_loss: 0.6338
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6698 - loss: 0.8053 - val_accuracy: 0.80
00 - val_loss: 0.6617
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6359 - loss: 0.8684 - val_accuracy: 0.7
692 - val_loss: 0.6806
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6428 - loss: 0.8665 - val_accuracy: 0.76
92 - val_loss: 0.6692
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5818 - loss: 0.9735 - val_accuracy: 0.78
46 - val_loss: 0.6845
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6365 - loss: 0.8470 - val_accuracy: 0.7
538 - val_loss: 0.7063
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6521 - loss: 0.8769 - val_accuracy: 0.6
615 - val_loss: 0.7286
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6425 - loss: 0.8565 - val_accuracy: 0.76
92 - val_loss: 0.6673
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5959 - loss: 0.9097 - val_accuracy: 0.78
46 - val_loss: 0.6377
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6601 - loss: 0.7891 - val_accuracy: 0.78
46 - val_loss: 0.6435
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6115 - loss: 0.8910 - val_accuracy: 0.64
62 - val_loss: 0.6942
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6011 - loss: 0.8949 - val_accuracy: 0.72
31 - val_loss: 0.6609
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7022 - loss: 0.7847 - val_accuracy: 0.80
00 - val_loss: 0.6324
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6460 - loss: 0.8193 - val_accuracy: 0.78
46 - val_loss: 0.6266
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6447 - loss: 0.7881 - val_accuracy: 0.78
46 - val_loss: 0.6395
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5987 - loss: 0.8675 - val_accuracy: 0.80
00 - val_loss: 0.6503
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6570 - loss: 0.7940 - val_accuracy: 0.80
00 - val_loss: 0.6449
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6192 - loss: 0.8635 - val_accuracy: 0.78
46 - val_loss: 0.6475
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6621 - loss: 0.8095 - val_accuracy: 0.78
46 - val_loss: 0.6287
```

```
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5704 - loss: 0.9204 - val_accuracy: 0.80
00 - val_loss: 0.6134
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6193 - loss: 0.8730 - val_accuracy: 0.80
00 - val_loss: 0.6078
```

In [104]:

```
clmodel2.evaluate(X_test_norm, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8075 - loss: 0.6422
```

Out[104]:

```
[0.6386579275131226, 0.8024691343307495]
```

Observation:

- **We can observe from the results above that adding dropout in the input layer does not seem a good choice for this model. From the initial 96% accuracy, it became 80.75%. The loss also increased when I added the dropout. This means that the reducing the nodes further may result to a badly fitted model. we only have 5 predictors and less than a thousand data entries, this maybe the reason why dropout does not show its advantage in this model.**

Dropout layer in Hidden layer

- **Using dropout in hidden layers may affect the result in a different way than by using dropout in the input layers. The idea is the same, we regularize the model by dropping nodes in the hidden layer.**

In [71]:

```
def cl_dropout_model2():

  model = tf.keras.models.Sequential([
      #input layer
      tf.keras.layers.Input((5,)),
      #hidden layers
      tf.keras.layers.Dense(5,  kernel_initializer = "normal" ,
                           activation = "relu"),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(3,  kernel_initializer = "normal" ,
                           activation = "relu"),
      tf.keras.layers.Dropout(0.2),
      #output layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal",
                           activation = "softmax")
  ])

  model.compile(Adam(learning_rate = 0.01),
                loss = "categorical_crossentropy",
                metrics=["accuracy"]
                )
  return model

clmodel3 = cl_dropout_model2()

knowledgeDF_model3 = clmodel3.fit(X_train_norm, y_train,
                              validation_data = (X_val_norm, y_val) ,
                              epochs = 100, verbose = 1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.2623 - loss: 1.5969 - val_accuracy: 0.2
769 - val_loss: 1.5578
Epoch 2/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3481 - loss: 1.5478 - val_accuracy: 0.2
769 - val_loss: 1.5073
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3446 - loss: 1.4856 - val_accuracy: 0.27
69 - val_loss: 1.4479
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3416 - loss: 1.4222 - val_accuracy: 0.27
69 - val_loss: 1.3864
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3095 - loss: 1.3483 - val_accuracy: 0.27
69 - val_loss: 1.3065
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3690 - loss: 1.3017 - val_accuracy: 0.53
85 - val_loss: 1.2092
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4466 - loss: 1.2121 - val_accuracy: 0.55
38 - val_loss: 1.0975
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4222 - loss: 1.1357 - val_accuracy: 0.55
38 - val_loss: 0.9763
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4675 - loss: 1.0151 - val_accuracy: 0.61
54 - val_loss: 0.8746
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5306 - loss: 0.9704 - val_accuracy: 0.61
54 - val_loss: 0.7983
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4905 - loss: 0.9838 - val_accuracy: 0.70
77 - val_loss: 0.7433
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4650 - loss: 0.9812 - val_accuracy: 0.76
92 - val_loss: 0.7271
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5556 - loss: 0.8493 - val_accuracy: 0.83
08 - val_loss: 0.7090
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5805 - loss: 0.8723 - val_accuracy: 0.81
54 - val_loss: 0.6800
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5326 - loss: 0.8842 - val_accuracy: 0.78
46 - val_loss: 0.6472
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5259 - loss: 0.8703 - val_accuracy: 0.83
08 - val_loss: 0.6290
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5499 - loss: 0.8491 - val_accuracy: 0.86
15 - val_loss: 0.6170
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5972 - loss: 0.8282 - val_accuracy: 0.86
15 - val_loss: 0.5980
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5876 - loss: 0.8055 - val_accuracy: 0.86
15 - val_loss: 0.5644
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6170 - loss: 0.7492 - val_accuracy: 0.86
15 - val_loss: 0.5282
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6006 - loss: 0.7957 - val_accuracy: 0.86
15 - val_loss: 0.5182
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5083 - loss: 0.8711 - val_accuracy: 0.81
54 - val_loss: 0.5375
Epoch 23/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6015 - loss: 0.7911 - val_accuracy: 0.86
15 - val_loss: 0.4876
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6393 - loss: 0.7423 - val_accuracy: 0.86
15 - val_loss: 0.4619
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5807 - loss: 0.7839 - val_accuracy: 0.86
15 - val_loss: 0.4548
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6488 - loss: 0.7528 - val_accuracy: 0.86
```

```
15 - val_loss: 0.4521
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6101 - loss: 0.7175 - val_accuracy: 0.86
15 - val_loss: 0.4320
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6642 - loss: 0.6493 - val_accuracy: 0.87
69 - val_loss: 0.4196
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6648 - loss: 0.7065 - val_accuracy: 0.87
69 - val_loss: 0.4090
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6003 - loss: 0.7626 - val_accuracy: 0.87
69 - val_loss: 0.4229
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6008 - loss: 0.7312 - val_accuracy: 0.86
15 - val_loss: 0.4322
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6781 - loss: 0.6632 - val_accuracy: 0.86
15 - val_loss: 0.4230
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6274 - loss: 0.7070 - val_accuracy: 0.86
15 - val_loss: 0.4260
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6680 - loss: 0.6809 - val_accuracy: 0.86
15 - val_loss: 0.3977
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6513 - loss: 0.6379 - val_accuracy: 0.86
15 - val_loss: 0.3803
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6520 - loss: 0.6935 - val_accuracy: 0.87
69 - val_loss: 0.3804
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6114 - loss: 0.7632 - val_accuracy: 0.96
92 - val_loss: 0.3785
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6529 - loss: 0.6675 - val_accuracy: 0.87
69 - val_loss: 0.3567
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6654 - loss: 0.7110 - val_accuracy: 0.90
77 - val_loss: 0.3646
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6796 - loss: 0.7528 - val_accuracy: 0.86
15 - val_loss: 0.3642
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6937 - loss: 0.6715 - val_accuracy: 0.84
62 - val_loss: 0.3743
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6327 - loss: 0.8163 - val_accuracy: 0.86
15 - val_loss: 0.3839
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6377 - loss: 0.7408 - val_accuracy: 0.86
15 - val_loss: 0.3767
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6235 - loss: 0.7447 - val_accuracy: 0.86
15 - val_loss: 0.3847
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6943 - loss: 0.7259 - val_accuracy: 0.84
62 - val_loss: 0.4131
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6634 - loss: 0.7115 - val_accuracy: 0.86
15 - val_loss: 0.4129
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6535 - loss: 0.7822 - val_accuracy: 0.86
15 - val_loss: 0.4408
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7094 - loss: 0.6467 - val_accuracy: 0.86
15 - val_loss: 0.3595
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6619 - loss: 0.7144 - val_accuracy: 0.84
62 - val_loss: 0.3569
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6735 - loss: 0.7335 - val_accuracy: 0.86
15 - val_loss: 0.3622
```

```
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7054 - loss: 0.7491 - val_accuracy: 0.86
15 - val_loss: 0.3852
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7099 - loss: 0.6093 - val_accuracy: 0.84
62 - val_loss: 0.3763
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6591 - loss: 0.7108 - val_accuracy: 0.84
62 - val_loss: 0.3586
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7095 - loss: 0.6259 - val_accuracy: 0.8
462 - val_loss: 0.3500
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7102 - loss: 0.6402 - val_accuracy: 0.95
38 - val_loss: 0.3629
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6611 - loss: 0.6914 - val_accuracy: 0.8
462 - val_loss: 0.3756
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7038 - loss: 0.6384 - val_accuracy: 0.8
462 - val_loss: 0.3767
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6805 - loss: 0.6586 - val_accuracy: 0.84
62 - val_loss: 0.3707
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7214 - loss: 0.6120 - val_accuracy: 0.8
462 - val_loss: 0.3595
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7237 - loss: 0.6752 - val_accuracy: 0.8
462 - val_loss: 0.3667
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7026 - loss: 0.6808 - val_accuracy: 0.84
62 - val_loss: 0.3580
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6782 - loss: 0.7443 - val_accuracy: 0.8
923 - val_loss: 0.3554
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6992 - loss: 0.7236 - val_accuracy: 0.8
615 - val_loss: 0.3621
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6549 - loss: 0.6779 - val_accuracy: 0.8
615 - val_loss: 0.3686
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6454 - loss: 0.6969 - val_accuracy: 0.8
615 - val_loss: 0.3518
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6745 - loss: 0.7167 - val_accuracy: 0.8
615 - val_loss: 0.3573
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6715 - loss: 0.6367 - val_accuracy: 0.8
615 - val_loss: 0.3503
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7443 - loss: 0.5929 - val_accuracy: 0.8
615 - val_loss: 0.3355
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7546 - loss: 0.6311 - val_accuracy: 0.8
769 - val_loss: 0.3425
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7296 - loss: 0.6423 - val_accuracy: 0.8
615 - val_loss: 0.3460
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7204 - loss: 0.6637 - val_accuracy: 0.86
15 - val_loss: 0.3584
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6783 - loss: 0.6726 - val_accuracy: 0.8
615 - val_loss: 0.3375
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6858 - loss: 0.6828 - val_accuracy: 0.87
69 - val_loss: 0.3285
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7251 - loss: 0.6870 - val_accuracy: 0.87
69 - val_loss: 0.3037
Epoch 75/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6757 - loss: 0.6677 - val_accuracy: 0.87
69 - val_loss: 0.3057
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7220 - loss: 0.6535 - val_accuracy: 0.87
69 - val_loss: 0.3240
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7307 - loss: 0.6210 - val_accuracy: 0.87
69 - val_loss: 0.3426
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6476 - loss: 0.7606 - val_accuracy: 0.87
69 - val_loss: 0.3326
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6429 - loss: 0.6979 - val_accuracy: 0.87
69 - val_loss: 0.3253
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6586 - loss: 0.7048 - val_accuracy: 0.87
69 - val_loss: 0.3202
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6751 - loss: 0.6500 - val_accuracy: 0.87
69 - val_loss: 0.3264
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6868 - loss: 0.6604 - val_accuracy: 0.92
31 - val_loss: 0.3137
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6851 - loss: 0.7059 - val_accuracy: 0.87
69 - val_loss: 0.3223
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7116 - loss: 0.5932 - val_accuracy: 0.87
69 - val_loss: 0.3376
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6558 - loss: 0.6645 - val_accuracy: 0.86
15 - val_loss: 0.3207
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6986 - loss: 0.6506 - val_accuracy: 0.86
15 - val_loss: 0.3143
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7107 - loss: 0.7128 - val_accuracy: 0.86
15 - val_loss: 0.3339
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7374 - loss: 0.6033 - val_accuracy: 0.86
15 - val_loss: 0.3335
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6854 - loss: 0.6656 - val_accuracy: 0.84
62 - val_loss: 0.3339
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7162 - loss: 0.6192 - val_accuracy: 0.93
85 - val_loss: 0.3425
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7017 - loss: 0.6684 - val_accuracy: 0.84
62 - val_loss: 0.3465
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7552 - loss: 0.6298 - val_accuracy: 0.84
62 - val_loss: 0.3626
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7153 - loss: 0.5711 - val_accuracy: 0.84
62 - val_loss: 0.3744
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7953 - loss: 0.6083 - val_accuracy: 0.84
62 - val_loss: 0.3904
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7218 - loss: 0.6720 - val_accuracy: 0.84
62 - val_loss: 0.3368
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7255 - loss: 0.6738 - val_accuracy: 0.86
15 - val_loss: 0.3225
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7050 - loss: 0.6310 - val_accuracy: 0.86
15 - val_loss: 0.3053
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7183 - loss: 0.6480 - val_accuracy: 0.86
15 - val_loss: 0.3187
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7528 - loss: 0.6522 - val_accuracy: 0.86
```

```
15 - val_loss: 0.3330
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8071 - loss: 0.5317 - val_accuracy: 0.84
62 - val_loss: 0.3329
```

In [105]:

```
clmodel3.evaluate(X_test_norm, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8254 - loss: 0.3997
```

Out[105]:

```
[0.4033966064453125, 0.8148148059844971]
```

Observation:

- **As we can see from the results above, by using dropout in the hidden layer, we achieved a slight uplift to the accuracy and loss of the training, validation and testing sets.**
- **From the previous 80% accuracy and loss of 0.6 of the model that uses dropout in the visible layer, we now have an accuracy of 82% and loss of 0.4 in the model that uses dropout in the hidden layer. Although this is a massive downgrade from the accuracy and loss of the baseline model, we can still infer that using dropout in the hidden layer is better than using it at the visible layer.**

Time-Based Learning Rate Schedule

- **Time based learning rate schedule is very useful in machine learning if you are still unsure on what the best learning rate for your model is.**
- **This technique decreases the learning rate with a fixed value overtime and gradually as the number of epochs increases[10]. The SGD optimizer has a parameter called Decay where we can specify a value that we want to be used as a value to decrease the learning rate gradually.**

In [111]:

```
def time_based_model():

  model = tf.keras.models.Sequential([
      #input layer
      tf.keras.layers.Input((5,)),
      #hidden layers
      tf.keras.layers.Dense(5,  kernel_initializer = "normal" ,
                           activation = "relu"),
      tf.keras.layers.Dense(3,  kernel_initializer = "normal" ,
                           activation = "relu"),
      #output layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal",
                           activation = "softmax")
  ])
  epochs = 100
  learning_rate = 0.1
  decay_rate = learning_rate / epochs
  momentum = 0.8
  adam = Adam(
    learning_rate=learning_rate,
    ema_momentum = momentum,
    weight_decay=decay_rate,
  )
  model.compile(optimizer = adam,
                loss = "categorical_crossentropy",
                metrics=["accuracy"]
                )
  return model
```

```
tb_model = time_based_model()

tb_model_history = tb_model.fit(X_train_norm, y_train,
                                validation_data = (X_val_norm, y_val) ,
                                epochs = 100, verbose = 1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.2483 - loss: 1.5153 - val_accuracy: 0.4
769 - val_loss: 1.1313
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4397 - loss: 1.1076 - val_accuracy: 0.76
92 - val_loss: 0.7738
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7937 - loss: 0.7694 - val_accuracy: 0.80
00 - val_loss: 0.6477
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7802 - loss: 0.6449 - val_accuracy: 0.84
62 - val_loss: 0.5054
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7640 - loss: 0.5883 - val_accuracy: 0.84
62 - val_loss: 0.4734
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8465 - loss: 0.4767 - val_accuracy: 0.86
15 - val_loss: 0.4522
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8379 - loss: 0.4830 - val_accuracy: 0.8
308 - val_loss: 0.4732
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8674 - loss: 0.4201 - val_accuracy: 0.84
62 - val_loss: 0.4500
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8683 - loss: 0.3991 - val_accuracy: 0.84
62 - val_loss: 0.4307
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8426 - loss: 0.4427 - val_accuracy: 0.86
15 - val_loss: 0.3681
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8365 - loss: 0.4085 - val_accuracy: 0.84
62 - val_loss: 0.3745
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8621 - loss: 0.3574 - val_accuracy: 0.86
15 - val_loss: 0.3862
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8502 - loss: 0.3793 - val_accuracy: 0.84
62 - val_loss: 0.3454
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8527 - loss: 0.3402 - val_accuracy: 0.84
62 - val_loss: 0.3707
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8524 - loss: 0.3280 - val_accuracy: 0.84
62 - val_loss: 0.3254
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8447 - loss: 0.3207 - val_accuracy: 0.84
62 - val_loss: 0.2700
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8630 - loss: 0.2468 - val_accuracy: 0.81
54 - val_loss: 0.3903
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9092 - loss: 0.2828 - val_accuracy: 0.96
92 - val_loss: 0.2322
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9381 - loss: 0.2216 - val_accuracy: 0.89
23 - val_loss: 0.2887
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9385 - loss: 0.2363 - val_accuracy: 0.84
62 - val_loss: 0.3266
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8582 - loss: 0.3631 - val_accuracy: 0.92
31 - val_loss: 0.2096
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9395 - loss: 0.2361 - val_accuracy: 0.95
38 - val_loss: 0.1920
Epoch 23/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9568 - loss: 0.1973 - val_accuracy: 0.92
31 - val_loss: 0.2200
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9590 - loss: 0.1788 - val_accuracy: 0.87
69 - val_loss: 0.2966
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9006 - loss: 0.2241 - val_accuracy: 0.90
77 - val_loss: 0.2426
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9444 - loss: 0.1747 - val_accuracy: 0.90
77 - val_loss: 0.2461
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9499 - loss: 0.1741 - val_accuracy: 0.87
69 - val_loss: 0.3983
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9522 - loss: 0.1709 - val_accuracy: 0.93
85 - val_loss: 0.1670
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9305 - loss: 0.1750 - val_accuracy: 0.93
85 - val_loss: 0.1982
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9449 - loss: 0.1593 - val_accuracy: 0.92
31 - val_loss: 0.1950
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9068 - loss: 0.2742 - val_accuracy: 0.8
615 - val_loss: 0.3679
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7907 - loss: 0.5380 - val_accuracy: 0.7
385 - val_loss: 0.6570
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6889 - loss: 0.6464 - val_accuracy: 0.73
85 - val_loss: 0.4425
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7430 - loss: 0.3904 - val_accuracy: 0.8
462 - val_loss: 0.3079
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8669 - loss: 0.2879 - val_accuracy: 0.8
462 - val_loss: 0.3228
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8613 - loss: 0.2717 - val_accuracy: 0.84
62 - val_loss: 0.2926
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8175 - loss: 0.6754 - val_accuracy: 0.78
46 - val_loss: 0.3558
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7436 - loss: 0.4391 - val_accuracy: 0.6
615 - val_loss: 0.6157
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7413 - loss: 0.5382 - val_accuracy: 0.6
923 - val_loss: 0.4964
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8112 - loss: 0.3666 - val_accuracy: 0.8
154 - val_loss: 0.4360
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8403 - loss: 0.3505 - val_accuracy: 0.8
154 - val_loss: 0.4876
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8434 - loss: 0.3171 - val_accuracy: 0.81
54 - val_loss: 0.4121
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8348 - loss: 0.3568 - val_accuracy: 0.8
462 - val_loss: 0.3829
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8490 - loss: 0.3270 - val_accuracy: 0.8
462 - val_loss: 0.3728
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8762 - loss: 0.3056 - val_accuracy: 0.8
308 - val_loss: 0.3846
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8468 - loss: 0.2930 - val_accuracy: 0.8
615 - val_loss: 0.3265
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8726 - loss: 0.2866 - val_accuracy: 0.8
```

```
462 - val_loss: 0.3423
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8497 - loss: 0.3026 - val_accuracy: 0.8
462 - val_loss: 0.3230
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8747 - loss: 0.2400 - val_accuracy: 0.8
308 - val_loss: 0.3234
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8307 - loss: 0.2925 - val_accuracy: 0.81
54 - val_loss: 0.3136
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8560 - loss: 0.2407 - val_accuracy: 0.83
08 - val_loss: 0.3474
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9000 - loss: 0.2600 - val_accuracy: 0.84
62 - val_loss: 0.3495
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8844 - loss: 0.2615 - val_accuracy: 0.86
15 - val_loss: 0.3298
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8691 - loss: 0.2567 - val_accuracy: 0.69
23 - val_loss: 0.4366
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8099 - loss: 0.3104 - val_accuracy: 0.80
00 - val_loss: 0.3903
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8693 - loss: 0.2806 - val_accuracy: 0.81
54 - val_loss: 0.3312
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8749 - loss: 0.2394 - val_accuracy: 0.81
54 - val_loss: 0.3265
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8706 - loss: 0.2576 - val_accuracy: 0.86
15 - val_loss: 0.2895
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9206 - loss: 0.2210 - val_accuracy: 0.86
15 - val_loss: 0.3039
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9292 - loss: 0.2066 - val_accuracy: 0.86
15 - val_loss: 0.3228
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9206 - loss: 0.1997 - val_accuracy: 0.84
62 - val_loss: 0.3658
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8145 - loss: 0.3313 - val_accuracy: 0.69
23 - val_loss: 0.5238
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7662 - loss: 0.3751 - val_accuracy: 0.81
54 - val_loss: 0.4878
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8530 - loss: 0.2969 - val_accuracy: 0.81
54 - val_loss: 0.4024
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8861 - loss: 0.2398 - val_accuracy: 0.81
54 - val_loss: 0.4172
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8684 - loss: 0.2489 - val_accuracy: 0.81
54 - val_loss: 0.3893
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8648 - loss: 0.2598 - val_accuracy: 0.81
54 - val_loss: 0.3918
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8578 - loss: 0.2658 - val_accuracy: 0.83
08 - val_loss: 0.3999
Epoch 69/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8391 - loss: 0.2765 - val_accuracy: 0.83
08 - val_loss: 0.3459
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8627 - loss: 0.2526 - val_accuracy: 0.81
54 - val_loss: 0.3883
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8742 - loss: 0.2510 - val_accuracy: 0.81
54 - val_loss: 0.3333
Epoch 72/100
```

```
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8686 - loss: 0.2279 - val_accuracy: 0.81
54 - val_loss: 0.4002
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8567 - loss: 0.2527 - val_accuracy: 0.81
54 - val_loss: 0.3455
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8491 - loss: 0.2477 - val_accuracy: 0.83
08 - val_loss: 0.3681
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8811 - loss: 0.2220 - val_accuracy: 0.83
08 - val_loss: 0.3618
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8416 - loss: 0.2346 - val_accuracy: 0.81
54 - val_loss: 0.3247
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8737 - loss: 0.2265 - val_accuracy: 0.81
54 - val_loss: 0.3823
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8540 - loss: 0.2072 - val_accuracy: 0.80
00 - val_loss: 0.3582
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8493 - loss: 0.2353 - val_accuracy: 0.81
54 - val_loss: 0.3284
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9081 - loss: 0.2042 - val_accuracy: 0.80
00 - val_loss: 0.3835
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8832 - loss: 0.2197 - val_accuracy: 0.84
62 - val_loss: 0.3242
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8851 - loss: 0.2722 - val_accuracy: 0.78
46 - val_loss: 0.4463
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8838 - loss: 0.2308 - val_accuracy: 0.83
08 - val_loss: 0.3195
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9333 - loss: 0.1738 - val_accuracy: 0.81
54 - val_loss: 0.3913
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9354 - loss: 0.1655 - val_accuracy: 0.84
62 - val_loss: 0.3226
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9512 - loss: 0.1551 - val_accuracy: 0.87
69 - val_loss: 0.2648
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9554 - loss: 0.1481 - val_accuracy: 0.89
23 - val_loss: 0.2673
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9642 - loss: 0.1394 - val_accuracy: 0.89
23 - val_loss: 0.2359
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9620 - loss: 0.1484 - val_accuracy: 0.90
77 - val_loss: 0.2566
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9550 - loss: 0.1372 - val_accuracy: 0.90
77 - val_loss: 0.2464
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9759 - loss: 0.1073 - val_accuracy: 0.90
77 - val_loss: 0.2504
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9594 - loss: 0.1393 - val_accuracy: 0.90
77 - val_loss: 0.2734
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9765 - loss: 0.1066 - val_accuracy: 0.90
77 - val_loss: 0.2323
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9810 - loss: 0.0936 - val_accuracy: 0.89
23 - val_loss: 0.2531
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9439 - loss: 0.1404 - val_accuracy: 0.92
31 - val_loss: 0.2067
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9570 - loss: 0.1535 - val_accuracy: 0.92
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9579 - loss: 0.1525 - val_accuracy: 0.92
31 - val_loss: 0.2192
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9764 - loss: 0.0790 - val_accuracy: 0.89
23 - val_loss: 0.2573
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9718 - loss: 0.1368 - val_accuracy: 0.87
69 - val_loss: 0.3208
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9717 - loss: 0.0856 - val_accuracy: 0.90
77 - val_loss: 0.2351
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9829 - loss: 0.0919 - val_accuracy: 0.92
31 - val_loss: 0.2729
```

In [112]:

```
tb_model.evaluate(X_test_norm, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9441 - loss: 0.2384
```

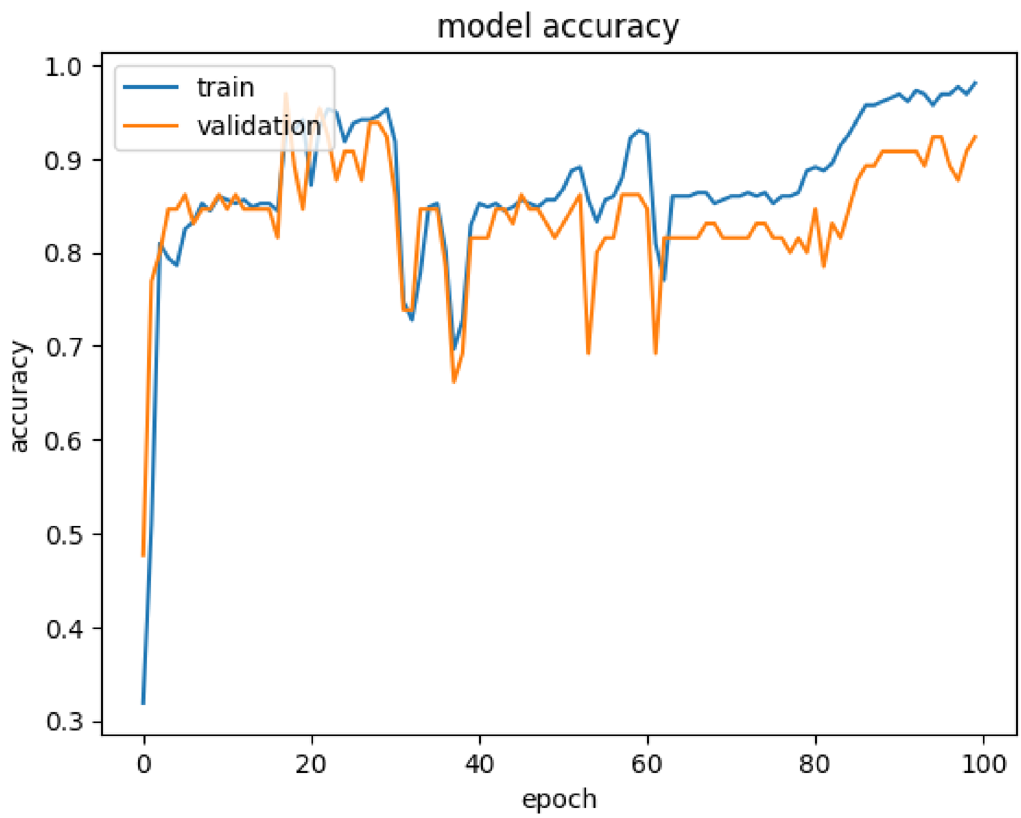
Out[112]:

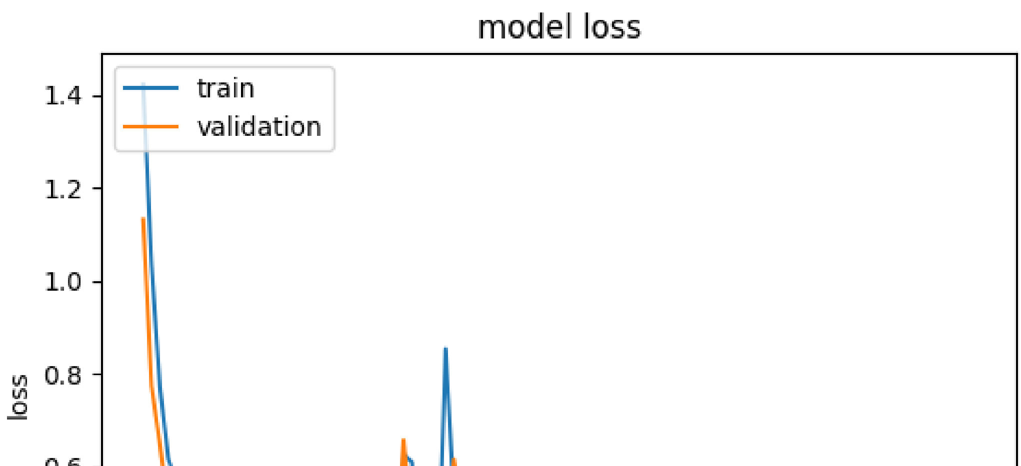
```
[0.23680146038532257, 0.9506173133850098]
```

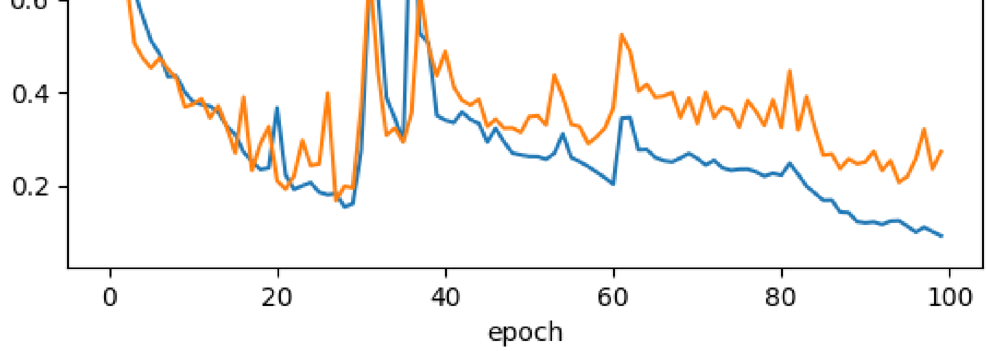
In [127]:

```
visualize_history(tb_model_history)
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







Observation:

- As we can see from above, the accuracy and loss of training and validation is pretty high and can be compared with our baseline model. The accuracy of our baseline model is 96% while this model that was configured with time based learning rate has an accuracy of 95% which is pretty close.
- There is a formula that was used for the decay, which is the decay = learning_rate/epochs. This is the value for every decrease of the initial learning rate for our model. It can be seen that for the first parts of the model training, the accuracy is low and it increased gradually as the value of learning rate decreased.
- Overall, using time-based learning rate is a good technique especially if you do not know what is the correct value for your learning rate. This can help you save time in changing the learning rate every now and then as an attempt to find the best learning rate manually.

Drop Based Learning Rate

- This learning rate schedule technique differs from Time-Based Learning Rate in a way it drops the learning rate. For the time-based learning rate, it decreases the learning rate in a continuous and consistent manner as the number of epochs increases. The drop based learning rate drops the learning rate in fixed intervals along the training epochs[10].
- This learning rate schedule technique is useful if you want to have a faster convergence to the gradient descent. Unlike the Time-Based learning rate that may take a long while and a large number of epochs, this technique can save you the time.

In [79]:

```
def step_decay(epoch):
 initial_lrate = 0.1
 drop = 0.5
 epochs_drop = 10.0
 lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
 return lrate
```

In [130]:

```
def drop_based_model():

  model = tf.keras.models.Sequential([
      #input layer
      tf.keras.layers.Input((5,)),
      tf.keras.layers.Dense(5, kernel_initializer = "normal",
                             activation = "relu"),
      #hidden layer
      tf.keras.layers.Dense(3,  kernel_initializer = "normal" ,
                             activation = "relu"),
      #output layer
      tf.keras.layers.Dense(5,  kernel_initializer = "normal",
                             activation = "softmax")
  ])

  model.compile(Adam(learning_rate = 0.0),
```

```
                loss = "categorical_crossentropy",
                metrics=["accuracy"]
                )
  return model

dbModel = drop_based_model()

lrate = LearningRateScheduler(step_decay)
callbacks_list = [lrate]

dbModel_history = dbModel.fit(X_train_norm, y_train, validation_data = (X_val_norm, y_val)
,
            epochs=100, callbacks=callbacks_list, verbose=1)
```

```
Epoch 1/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.3111 - loss: 1.4732 - val_accuracy: 0.5
231 - val_loss: 0.9601 - learning_rate: 0.1000
Epoch 2/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6011 - loss: 0.8114 - val_accuracy: 0.80
00 - val_loss: 0.4893 - learning_rate: 0.1000
Epoch 3/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7892 - loss: 0.5113 - val_accuracy: 0.89
23 - val_loss: 0.3688 - learning_rate: 0.1000
Epoch 4/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8038 - loss: 0.4103 - val_accuracy: 0.70
77 - val_loss: 0.7273 - learning_rate: 0.1000
Epoch 5/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7776 - loss: 0.4949 - val_accuracy: 0.89
23 - val_loss: 0.2796 - learning_rate: 0.1000
Epoch 6/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9222 - loss: 0.2996 - val_accuracy: 0.86
15 - val_loss: 0.2668 - learning_rate: 0.1000
Epoch 7/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8730 - loss: 0.3111 - val_accuracy: 0.80
00 - val_loss: 0.4374 - learning_rate: 0.1000
Epoch 8/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8742 - loss: 0.2649 - val_accuracy: 0.92
31 - val_loss: 0.2197 - learning_rate: 0.1000
Epoch 9/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9303 - loss: 0.1701 - val_accuracy: 0.93
85 - val_loss: 0.1479 - learning_rate: 0.1000
Epoch 10/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9709 - loss: 0.1235 - val_accuracy: 0.90
77 - val_loss: 0.1932 - learning_rate: 0.0500
Epoch 11/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9648 - loss: 0.1081 - val_accuracy: 0.92
31 - val_loss: 0.1738 - learning_rate: 0.0500
Epoch 12/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9795 - loss: 0.0739 - val_accuracy: 0.90
77 - val_loss: 0.1676 - learning_rate: 0.0500
Epoch 13/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9755 - loss: 0.0795 - val_accuracy: 0.92
31 - val_loss: 0.1489 - learning_rate: 0.0500
Epoch 14/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9359 - loss: 0.1580 - val_accuracy: 0.95
38 - val_loss: 0.1339 - learning_rate: 0.0500
Epoch 15/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9352 - loss: 0.1406 - val_accuracy: 0.90
77 - val_loss: 0.2398 - learning_rate: 0.0500
Epoch 16/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9729 - loss: 0.0870 - val_accuracy: 0.92
31 - val_loss: 0.1802 - learning_rate: 0.0500
Epoch 17/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9745 - loss: 0.0854 - val_accuracy: 0.92
31 - val_loss: 0.1823 - learning_rate: 0.0500
Epoch 18/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9915 - loss: 0.0745 - val_accuracy: 0.93
85 - val_loss: 0.1410 - learning_rate: 0.0500
Epoch 19/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9567 - loss: 0.1130 - val_accuracy: 0.93
85 - val_loss: 0.1760 - learning_rate: 0.0500
Epoch 20/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9730 - loss: 0.0819 - val_accuracy: 0.95
```

```
38 - val_loss: 0.1780 - learning_rate: 0.0250
Epoch 21/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9740 - loss: 0.0933 - val_accuracy: 0.93
85 - val_loss: 0.1803 - learning_rate: 0.0250
Epoch 22/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9756 - loss: 0.0668 - val_accuracy: 0.93
85 - val_loss: 0.1769 - learning_rate: 0.0250
Epoch 23/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9872 - loss: 0.0433 - val_accuracy: 0.93
85 - val_loss: 0.1855 - learning_rate: 0.0250
Epoch 24/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9747 - loss: 0.0610 - val_accuracy: 0.93
85 - val_loss: 0.1633 - learning_rate: 0.0250
Epoch 25/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9556 - loss: 0.0962 - val_accuracy: 0.9
538 - val_loss: 0.1243 - learning_rate: 0.0250
Epoch 26/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9626 - loss: 0.1137 - val_accuracy: 0.90
77 - val_loss: 0.2002 - learning_rate: 0.0250
Epoch 27/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9568 - loss: 0.0967 - val_accuracy: 0.93
85 - val_loss: 0.1772 - learning_rate: 0.0250
Epoch 28/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9723 - loss: 0.0725 - val_accuracy: 0.95
38 - val_loss: 0.1574 - learning_rate: 0.0250
Epoch 29/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9797 - loss: 0.0597 - val_accuracy: 0.95
38 - val_loss: 0.1651 - learning_rate: 0.0250
Epoch 30/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9703 - loss: 0.0771 - val_accuracy: 0.93
85 - val_loss: 0.1558 - learning_rate: 0.0125
Epoch 31/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9765 - loss: 0.0780 - val_accuracy: 0.93
85 - val_loss: 0.1458 - learning_rate: 0.0125
Epoch 32/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9803 - loss: 0.0529 - val_accuracy: 0.92
31 - val_loss: 0.1405 - learning_rate: 0.0125
Epoch 33/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9798 - loss: 0.0556 - val_accuracy: 0.93
85 - val_loss: 0.1550 - learning_rate: 0.0125
Epoch 34/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9922 - loss: 0.0408 - val_accuracy: 0.93
85 - val_loss: 0.1605 - learning_rate: 0.0125
Epoch 35/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9871 - loss: 0.0549 - val_accuracy: 0.93
85 - val_loss: 0.1532 - learning_rate: 0.0125
Epoch 36/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9876 - loss: 0.0471 - val_accuracy: 0.93
85 - val_loss: 0.1577 - learning_rate: 0.0125
Epoch 37/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9911 - loss: 0.0455 - val_accuracy: 0.93
85 - val_loss: 0.1555 - learning_rate: 0.0125
Epoch 38/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9925 - loss: 0.0453 - val_accuracy: 0.95
38 - val_loss: 0.1517 - learning_rate: 0.0125
Epoch 39/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9905 - loss: 0.0444 - val_accuracy: 0.92
31 - val_loss: 0.1561 - learning_rate: 0.0125
Epoch 40/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9897 - loss: 0.0462 - val_accuracy: 0.92
31 - val_loss: 0.1607 - learning_rate: 0.0063
Epoch 41/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9959 - loss: 0.0412 - val_accuracy: 0.92
31 - val_loss: 0.1489 - learning_rate: 0.0063
Epoch 42/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9795 - loss: 0.0645 - val_accuracy: 0.92
31 - val_loss: 0.1591 - learning_rate: 0.0063
Epoch 43/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9490 - loss: 0.1440 - val_accuracy: 0.90
77 - val_loss: 0.1572 - learning_rate: 0.0063
Epoch 44/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9672 - loss: 0.0863 - val_accuracy: 0.92
31 - val_loss: 0.1680 - learning_rate: 0.0063
Epoch 45/100
```

```
Epoch 45/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9871 - loss: 0.0725 - val_accuracy: 0.93
85 - val_loss: 0.1784 - learning_rate: 0.0063
Epoch 46/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9822 - loss: 0.0462 - val_accuracy: 0.95
38 - val_loss: 0.1730 - learning_rate: 0.0063
Epoch 47/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9895 - loss: 0.0450 - val_accuracy: 0.95
38 - val_loss: 0.1558 - learning_rate: 0.0063
Epoch 48/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9815 - loss: 0.0632 - val_accuracy: 0.95
38 - val_loss: 0.1488 - learning_rate: 0.0063
Epoch 49/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9811 - loss: 0.0538 - val_accuracy: 0.95
38 - val_loss: 0.1543 - learning_rate: 0.0063
Epoch 50/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9891 - loss: 0.0422 - val_accuracy: 0.93
85 - val_loss: 0.1600 - learning_rate: 0.0031
Epoch 51/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9887 - loss: 0.0473 - val_accuracy: 0.93
85 - val_loss: 0.1614 - learning_rate: 0.0031
Epoch 52/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9892 - loss: 0.0483 - val_accuracy: 0.93
85 - val_loss: 0.1620 - learning_rate: 0.0031
Epoch 53/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9826 - loss: 0.0615 - val_accuracy: 0.9
385 - val_loss: 0.1624 - learning_rate: 0.0031
Epoch 54/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9830 - loss: 0.0455 - val_accuracy: 0.9
385 - val_loss: 0.1580 - learning_rate: 0.0031
Epoch 55/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9917 - loss: 0.0370 - val_accuracy: 0.9
385 - val_loss: 0.1576 - learning_rate: 0.0031
Epoch 56/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9856 - loss: 0.0454 - val_accuracy: 0.93
85 - val_loss: 0.1611 - learning_rate: 0.0031
Epoch 57/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9810 - loss: 0.0492 - val_accuracy: 0.93
85 - val_loss: 0.1622 - learning_rate: 0.0031
Epoch 58/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9840 - loss: 0.0513 - val_accuracy: 0.93
85 - val_loss: 0.1672 - learning_rate: 0.0031
Epoch 59/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9783 - loss: 0.0501 - val_accuracy: 0.93
85 - val_loss: 0.1697 - learning_rate: 0.0031
Epoch 60/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9871 - loss: 0.0365 - val_accuracy: 0.93
85 - val_loss: 0.1680 - learning_rate: 0.0016
Epoch 61/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9856 - loss: 0.0550 - val_accuracy: 0.9
385 - val_loss: 0.1665 - learning_rate: 0.0016
Epoch 62/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9858 - loss: 0.0480 - val_accuracy: 0.9
385 - val_loss: 0.1646 - learning_rate: 0.0016
Epoch 63/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9939 - loss: 0.0335 - val_accuracy: 0.9
385 - val_loss: 0.1650 - learning_rate: 0.0016
Epoch 64/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9922 - loss: 0.0353 - val_accuracy: 0.9
385 - val_loss: 0.1643 - learning_rate: 0.0016
Epoch 65/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9903 - loss: 0.0412 - val_accuracy: 0.9
385 - val_loss: 0.1597 - learning_rate: 0.0016
Epoch 66/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9852 - loss: 0.0528 - val_accuracy: 0.93
85 - val_loss: 0.1584 - learning_rate: 0.0016
Epoch 67/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9915 - loss: 0.0422 - val_accuracy: 0.93
85 - val_loss: 0.1584 - learning_rate: 0.0016
Epoch 68/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9899 - loss: 0.0440 - val_accuracy: 0.9
385 - val_loss: 0.1574 - learning_rate: 0.0016
Epoch 69/100
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9847 - loss: 0.0477 - val_accuracy: 0.9
385 - val_loss: 0.1574 - learning_rate: 0.0016
Epoch 70/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9857 - loss: 0.0486 - val_accuracy: 0.9
385 - val_loss: 0.1576 - learning_rate: 7.8125e-04
Epoch 71/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9850 - loss: 0.0599 - val_accuracy: 0.9
385 - val_loss: 0.1575 - learning_rate: 7.8125e-04
Epoch 72/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9945 - loss: 0.0332 - val_accuracy: 0.9
385 - val_loss: 0.1573 - learning_rate: 7.8125e-04
Epoch 73/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9915 - loss: 0.0409 - val_accuracy: 0.93
85 - val_loss: 0.1571 - learning_rate: 7.8125e-04
Epoch 74/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9864 - loss: 0.0443 - val_accuracy: 0.93
85 - val_loss: 0.1581 - learning_rate: 7.8125e-04
Epoch 75/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9830 - loss: 0.0571 - val_accuracy: 0.93
85 - val_loss: 0.1578 - learning_rate: 7.8125e-04
Epoch 76/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9951 - loss: 0.0312 - val_accuracy: 0.93
85 - val_loss: 0.1577 - learning_rate: 7.8125e-04
Epoch 77/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9887 - loss: 0.0413 - val_accuracy: 0.93
85 - val_loss: 0.1583 - learning_rate: 7.8125e-04
Epoch 78/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9891 - loss: 0.0480 - val_accuracy: 0.93
85 - val_loss: 0.1581 - learning_rate: 7.8125e-04
Epoch 79/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9834 - loss: 0.0480 - val_accuracy: 0.9
385 - val_loss: 0.1587 - learning_rate: 7.8125e-04
Epoch 80/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9860 - loss: 0.0544 - val_accuracy: 0.9
385 - val_loss: 0.1585 - learning_rate: 3.9063e-04
Epoch 81/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9891 - loss: 0.0370 - val_accuracy: 0.9
385 - val_loss: 0.1588 - learning_rate: 3.9063e-04
Epoch 82/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9915 - loss: 0.0370 - val_accuracy: 0.9
385 - val_loss: 0.1590 - learning_rate: 3.9063e-04
Epoch 83/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9834 - loss: 0.0542 - val_accuracy: 0.9
385 - val_loss: 0.1589 - learning_rate: 3.9063e-04
Epoch 84/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9909 - loss: 0.0403 - val_accuracy: 0.9
385 - val_loss: 0.1600 - learning_rate: 3.9063e-04
Epoch 85/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9943 - loss: 0.0386 - val_accuracy: 0.9
231 - val_loss: 0.1677 - learning_rate: 3.9063e-04
Epoch 86/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9945 - loss: 0.0389 - val_accuracy: 0.9
231 - val_loss: 0.1704 - learning_rate: 3.9063e-04
Epoch 87/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9943 - loss: 0.0451 - val_accuracy: 0.9
231 - val_loss: 0.1714 - learning_rate: 3.9063e-04
Epoch 88/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9942 - loss: 0.0405 - val_accuracy: 0.9
231 - val_loss: 0.1711 - learning_rate: 3.9063e-04
Epoch 89/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9962 - loss: 0.0362 - val_accuracy: 0.92
31 - val_loss: 0.1710 - learning_rate: 3.9063e-04
Epoch 90/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9938 - loss: 0.0367 - val_accuracy: 0.92
31 - val_loss: 0.1709 - learning_rate: 1.9531e-04
Epoch 91/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9927 - loss: 0.0450 - val_accuracy: 0.92
31 - val_loss: 0.1704 - learning_rate: 1.9531e-04
Epoch 92/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9933 - loss: 0.0443 - val_accuracy: 0.92
31 - val_loss: 0.1701 - learning_rate: 1.9531e-04
Epoch 93/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9938 - loss: 0.0441 - val_accuracy: 0.92
31 - val_loss: 0.1697 - learning_rate: 1.9531e-04
```

```
31 - val_loss: 0.1697 - learning_rate: 1.9531e-04
Epoch 94/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9861 - loss: 0.0548 - val_accuracy: 0.92
31 - val_loss: 0.1694 - learning_rate: 1.9531e-04
Epoch 95/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9861 - loss: 0.0594 - val_accuracy: 0.92
31 - val_loss: 0.1690 - learning_rate: 1.9531e-04
Epoch 96/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9896 - loss: 0.0503 - val_accuracy: 0.92
31 - val_loss: 0.1687 - learning_rate: 1.9531e-04
Epoch 97/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9903 - loss: 0.0481 - val_accuracy: 0.92
31 - val_loss: 0.1685 - learning_rate: 1.9531e-04
Epoch 98/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9891 - loss: 0.0509 - val_accuracy: 0.92
31 - val_loss: 0.1681 - learning_rate: 1.9531e-04
Epoch 99/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9956 - loss: 0.0339 - val_accuracy: 0.92
31 - val_loss: 0.1673 - learning_rate: 1.9531e-04
Epoch 100/100
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9961 - loss: 0.0380 - val_accuracy: 0.92
31 - val_loss: 0.1670 - learning_rate: 9.7656e-05
```

In [131]:

```
dbModel.evaluate(X_test_norm, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9519 - loss: 0.1744
```

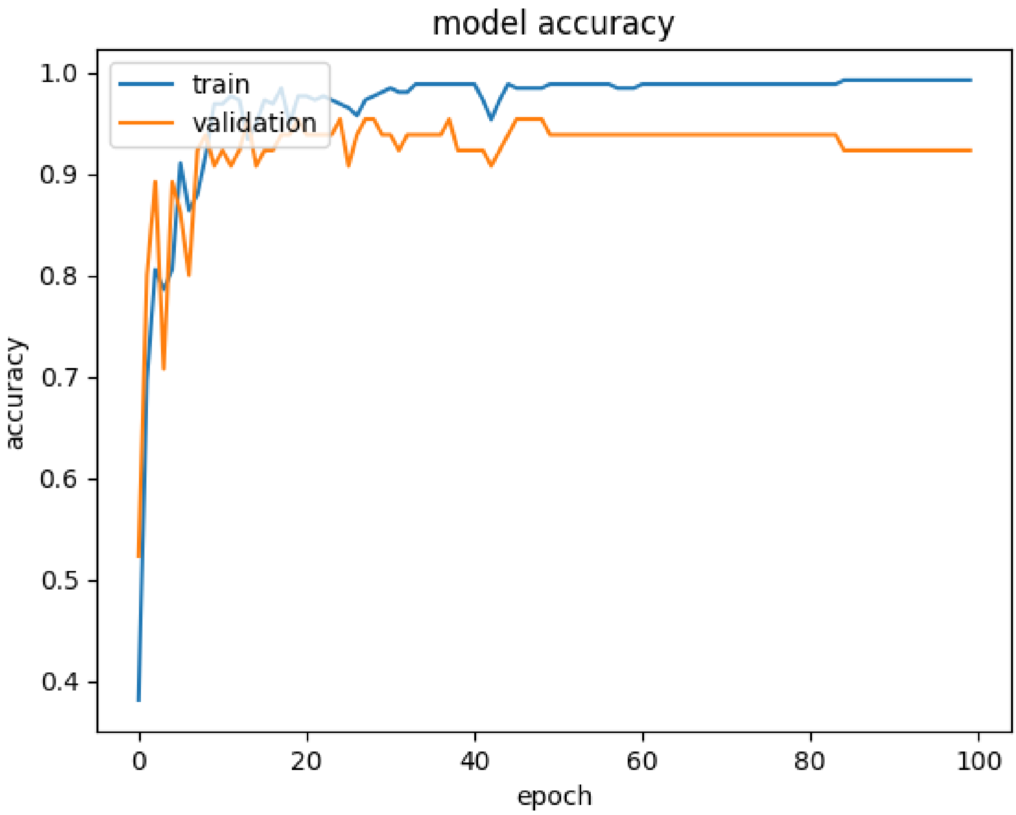
Out[131]:

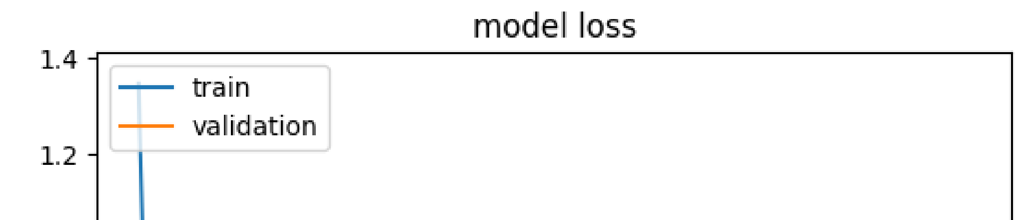
```
[0.16233216226100922, 0.9506173133850098]
```

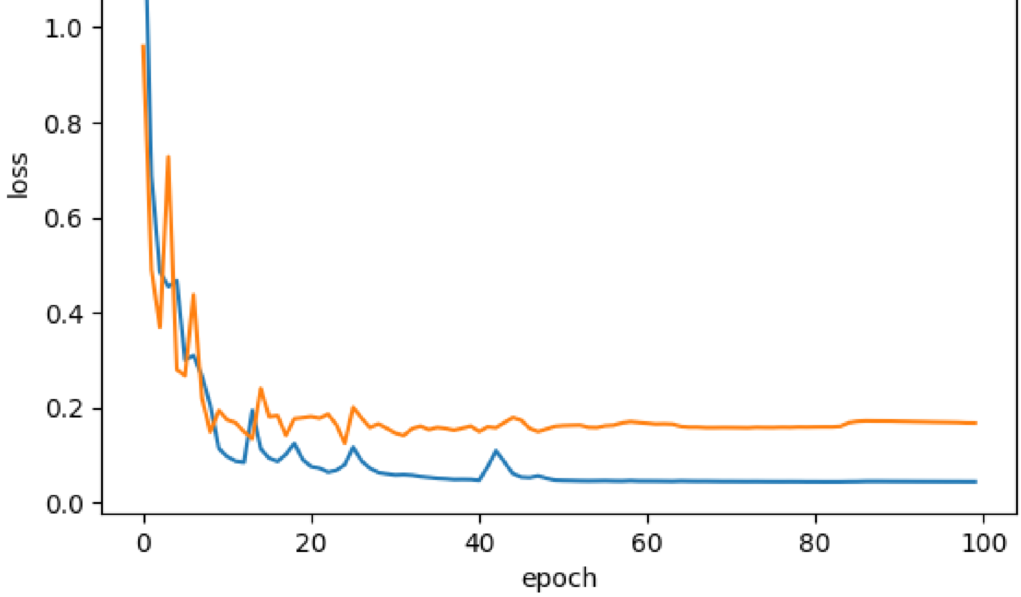
In [133]:

```
visualize_history(dbModel_history)
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss', 'learning_rate'])
```







Observation:

- We can observe here that the fluctuations are lesser than the previous Time Based learning rate schedule. This is because the learning rate only decreases every 10 epochs and it decreases a fix amount of 0.5. The training of a model will be much easier with this technique since it can help you find the best learning rate for your model. The only downside is that if there is a large step for the decrease of your learning rate, the accuracy and loss may fluctuate and your model can jump over the local minima of the gradient descent.

Summary and Conclusion:

Summary:

- For this activity, I used the User Knowledge Domain regarding DC Electrical motors as the dataset, and it has 5 predictors and 1 target variable. The model for this is a multiclass classification model since the label or the target variabel consists of 4 values. I converted the 4 values to integer and used one hot encoding to turn them into categorical variables. Then I trained the base model and got a pretty high accuracy and low loss. By using this model as a baseline, I compared every model after to it and listed some advantages and disadvantages for every task that I finish.
- I accomplished all the task listed in the activity and provided some insightful observations and listed some references to backup the claims that I have made. The main tasks for this activity are saving models, using dropout regularization and using learning rate schedules.
- I successfully saved models in different formats such as HDF5, JSON, and YAML. I also learned their differences and their advantages and disadvantages which are listed in my observations. I also showed the application of the checkpoints to the model and how it can save you time from training your model again and agian. Visualizing the result of your model training is also important in inferring useful information from your training and validation accuracy and loss.
- For the dropout regularization, I accomplished the task but found out that dropout is not always good and it may depend on the dataset that you are using. In my case, I only have 5 predictors and 403 data entries. There is only one predictor with a very high correlation with the target variable, so if there is an iteration where this predictor was dropped by the dropout layer, it can affect the accuracy of this model.
- Lastly, for the learning rate schedules, I also accomplished them and found out that this is a very useful technique in machine learning. Time Based learning rate schedule is very useful if you want to have a consistent gradual decrease of learning rate. This learning rate can find the local minima easily but it may take a time. For the drop based learning rate, this is faster than the time based learning rate but it relies on the step and the number of epochs before decreasing the learning rate. These factors may affect the results and may cause the learning rate to pass over the local minima. But if this is utilized right, this can save you more time than the time based learning rate and also memories while training the model.

Conclusion:

- Overall, this activity filled the gaps in my knowledge regarding the different techniques that can be utilize to make machine learning much easier and bearable. It also introduced me to new concept and it gave me new ideas how to make my life easier the next time I am doing a machine learning project. It also mentioned some tips and tricks which I can use in the future if I ever found myself stuck in a certain part of my machine learning project. This activity also taught me how to do extensive research and how important the understanding of the different concepts in machine learning can affect your final output. I am very happy I can build my foundation in machine learning and I am looking forward to more activities like this. I will certainly recommend this activity to future students enrolling in this course so that they can build their foundation in machine learning and learn some techniques to make their life easier.

REFERENCES:

[1]IBM, "What Is Exploratory Data Analysis? | IBM," www.ibm.com, 2020. https://www.ibm.com/topics/exploratory-data-analysis

[2]"Data Dependencies | Machine Learning," Google for Developers. https://developers.google.com/machine-learning/crash-course/data-dependencies/video-lecture

[3]I. Guyon, "A scaling law for the validation-set training-set size ratio." Accessed: Nov. 05, 2023. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=452e6c05d46e061290fefff8b46d0ff161998677

[4]"Splitting into train, dev and test sets," cs230.stanford.edu. https://cs230.stanford.edu/blog/split/

[5]F. Chollet, "The Sequential model," keras.io, Apr. 12, 2020. https://keras.io/guides/sequential_model/

[6]J. Brownlee, "How to Choose Loss Functions When Training Deep Learning Neural Networks," Machine Learning Mastery, Aug. 15, 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[7]"HDF5 for Python — h5py 3.7.0 documentation," docs.h5py.org. https://docs.h5py.org/en/stable/

[8]J. Brownlee, "How to Save and Load Your Keras Deep Learning Model," Machine Learning Mastery, May 12, 2019. https://machinelearningmastery.com/save-load-keras-deep-learning-models/

[9]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, and Y. Bengio, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014, Available: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[10]J. Brownlee, "Using Learning Rate Schedules for Deep Learning Models in Python with Keras," Machine Learning Mastery, Jun. 21, 2016. https://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/

Link for Dataset

- Kahraman,Hamdi, Colak,Ilhami, and Sagiroglu,Seref. (2013). User Knowledge Modeling. UCI Machine Learning Repository. https://doi.org/10.24432/C5231X.